Report Tecnico

Aloia, Arena, Nasso, Saporito

Contents

1 Setup dell'ambiente

Per le simulazioni sono state utilizzate le seguenti tecnologie:

- Sistema operativo: `Ubuntu 22.04 LTS`
- Ambiente di simulazione: `OMNeT++ 6.0.3`
- Framework di rete: `INET 4.5`
- Libreria per algebra lineare: `Eigen 3.4.0`
- Libreria crittografica per EDDSA: `Libsodium 1.0.18`
- Libreria crittografica per ECDSA: `OpenSSL 3.0.2`

1.1 Installazione delle dipendenze

1.1.1 Eigen 3.4.0

Per installare la libreria Eigen su Ubuntu 22.04:

```
sudo apt-get update

sudo apt-get install libeigen3-dev
```

Verificare la versione installata con:

```
dpkg -s libeigen3-dev | grep Version
```

1.1.2 Libsodium 1.0.18

Per installare la libreria libsodium:

```
sudo apt-get update

sudo apt-get install libsodium-dev
```

Verificare la versione con:

```
pkg-config --modversion libsodium
```

1.1.3 OpenSSL 3.0.2

OpenSSL 3.0.2 è incluso in Ubuntu 22.04. Installazione/verifica:

```
sudo apt-get install openssl libssl-dev
```

Verificare la versione con:

```
openssl version
```

Nota: Tutte le versioni indicate sono quelle predefinite dei repository ufficiali di Ubuntu 22.04 LTS. Per installazioni personalizzate o versioni specifiche, consultare la documentazione ufficiale di ciascun pacchetto.

1.1.4 Installazione di Omnet++ 6.0.3

Scaricare l'archivio TGZ di Omnet++ 6.0.3 da `https://omnetpp.org/download/old`

Installazione delle dipendenze

```
sudo apt-get update

sudo apt-get install build-essential clang lld gdb bison flex perl \
    python3 python3-pip qtbase5-dev qtchooser qt5-qmake qtbase5-dev-tools \
    libqt5opengl5-dev libxml2-dev zlib1g-dev doxygen graphviz \
    libwebkit2gtk-4.0-37 xdg-utils

python3 -m pip install --user --upgrade numpy pandas matplotlib scipy \
    seaborn posix_ipc

cd && nano .bashrc          # o .zshrc

# aggiungere la riga che porta la path di installazione delle librerie
# python in modalità user, nel caso della VM
# export PATH="/home/user/.local/bin:$PATH"

sudo apt-get install libopenscenegraph-dev

sudo apt-get install mpi-default-dev

sudo nano /etc/sysctl.d/10-ptrace.conf

# cambiare la riga kernel.yama.ptrace_scope = 1 in kernel.yama.ptrace_scope = 0
```

Installazione Dall'estrazione dell'archivio TGZ risulterà una cartella omnetpp-6.0.3, posizionarla dove si preferisce, a patto che, ove sia esplicitato (nei comandi o negli script usati), poi ci si ricordi di modificare il path. Nella VM fornita è stata posizionata nella home dell'utente.

```
cd /home/user/omnetpp-6.0.3

source setenv

./configure

make
```

Primo avvio Lanciare il comando:

```
omnetpp
```

Selezionare il workspace

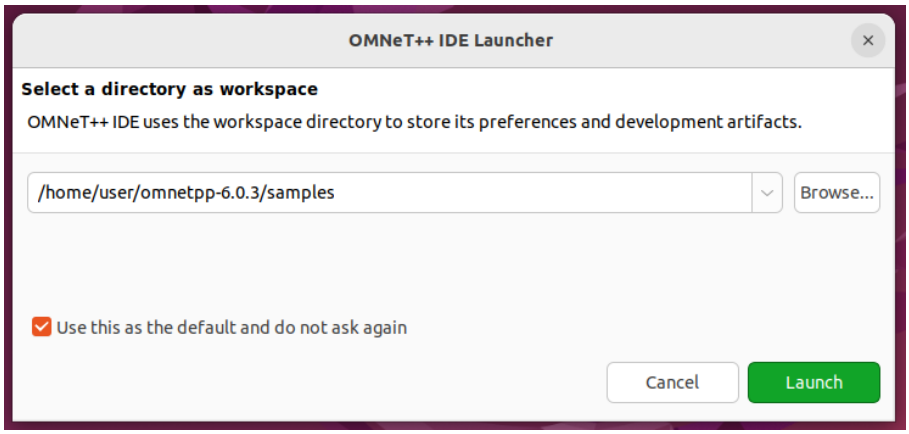


Figure 1: Setup workspace Omnet++

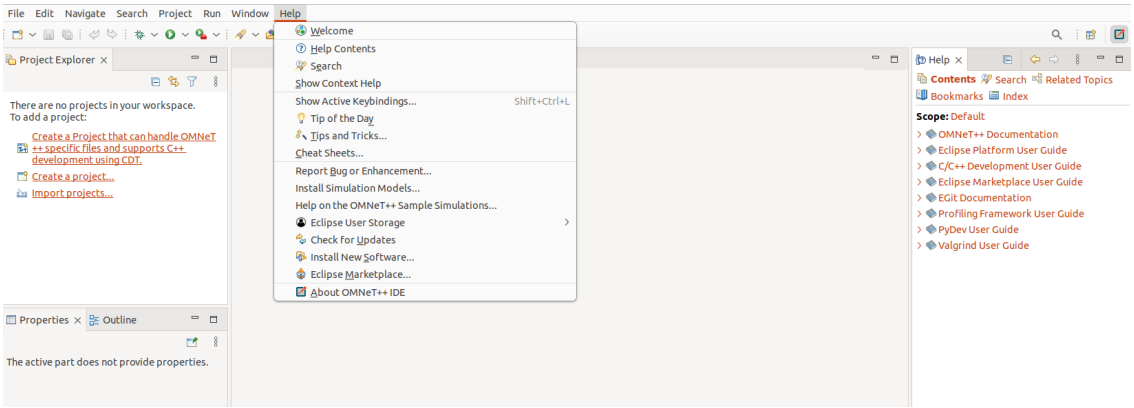
Dopo il primo avvio non selezionare nulla da installare. Lo si fa manualmente nei passi successivi.

Avvio standard Lanciare il comando:

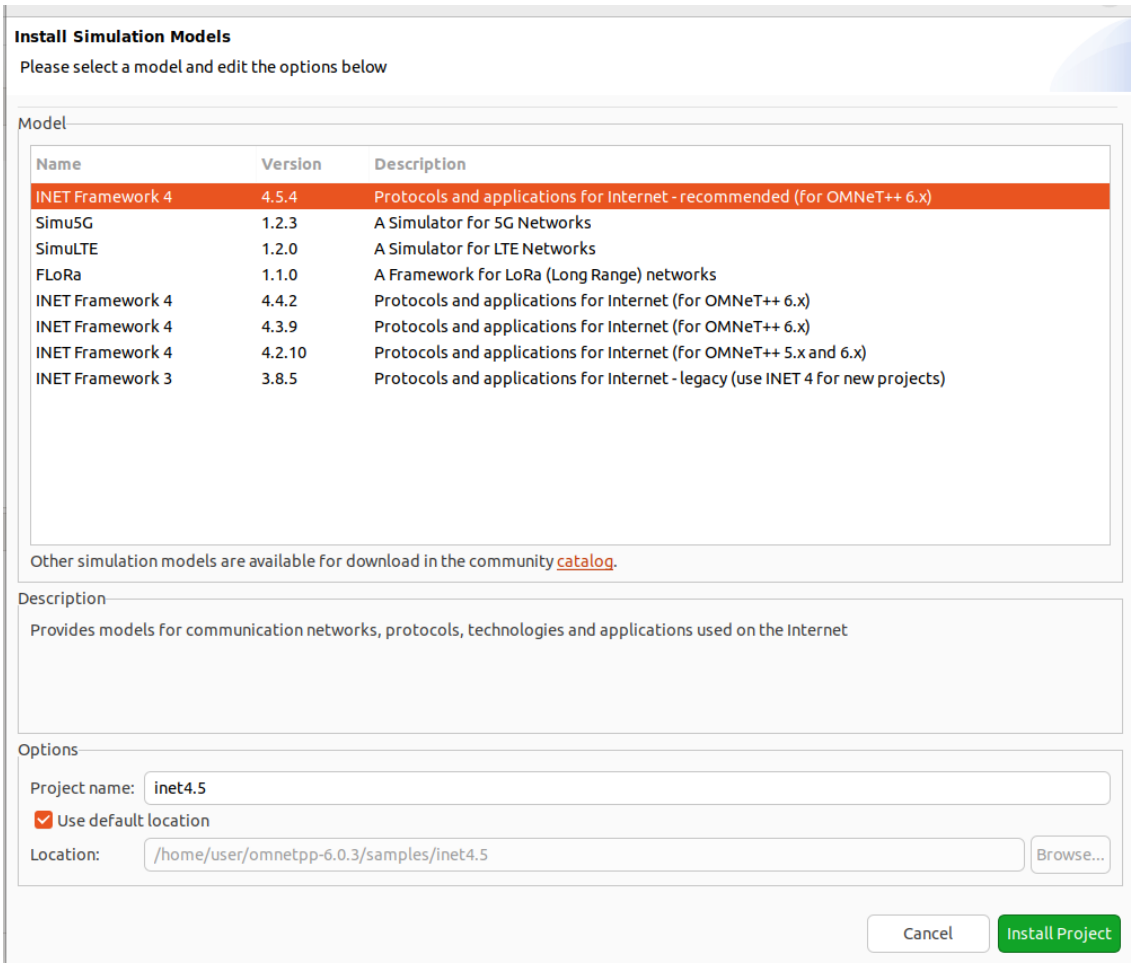
```
cd && cd omnetpp-6.0.3 && source setenv && omnetpp
```

1.1.5 Installazione di Inet 4.5

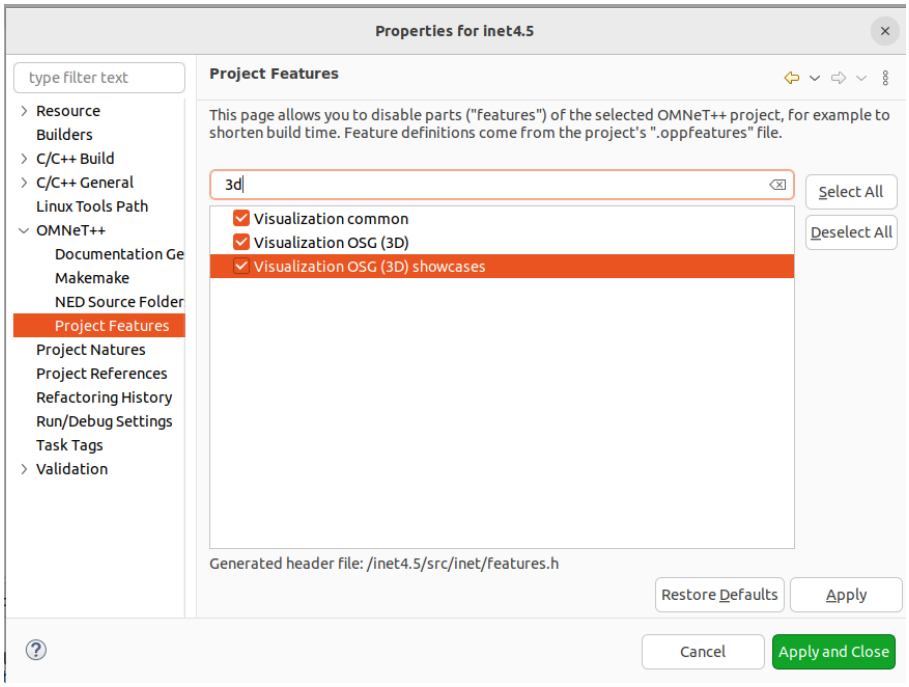
Cliccare su *Help* nella barra in alto e poi cliccare su *Install Simulation Models...*



Selezionare *Inet 4.5* e cliccare su *Install Project*

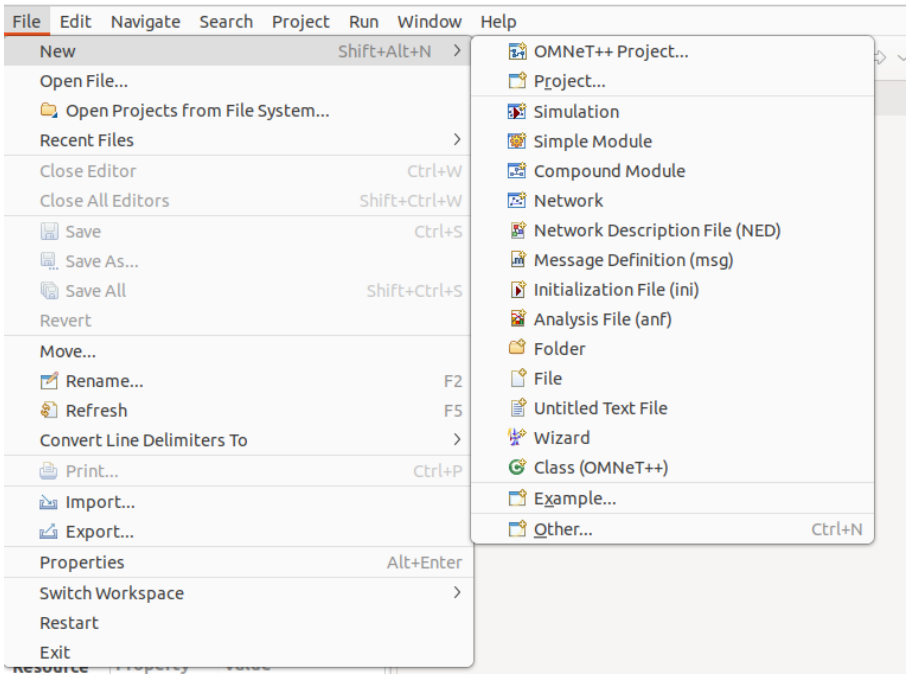


Dopo il completamento dell'installazione, tasto destro su *inet4.5* → *Properties* e abilitare il *3D* come segue: selezionare *Visualization OSG (3D)* e *Visualization OSG (3D) showcases* in *Project Features* e poi *Apply and Close*.

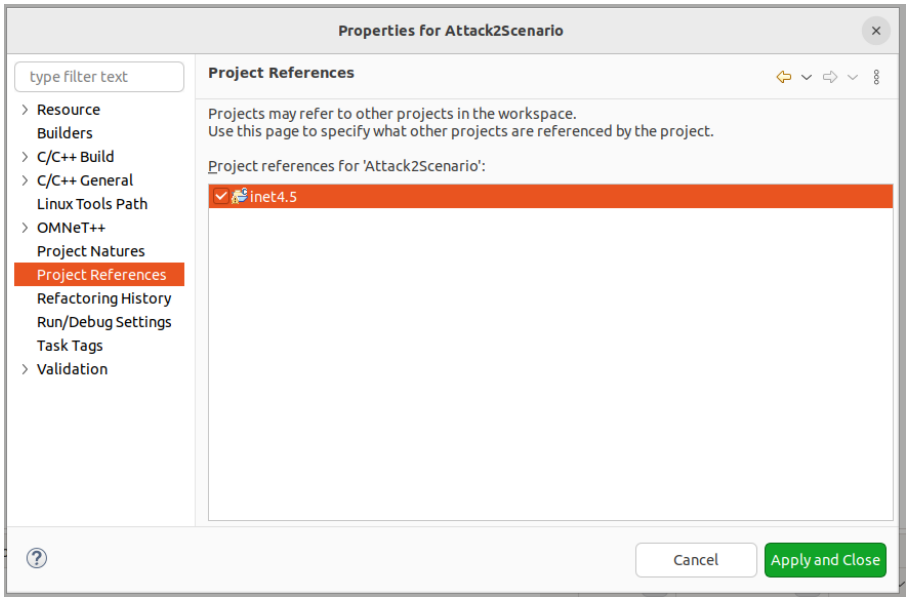


1.1.6 Setup progetti

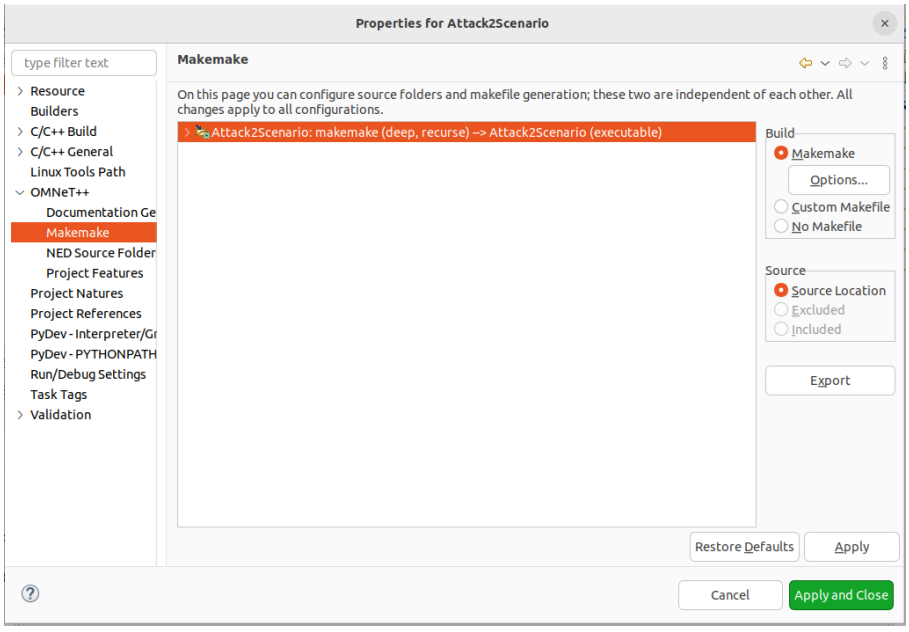
Nella VM tutti i progetti sono stati già posizionati nel workspace, ad ogni modo, se li si vuole importare, bisogna creare un nuovo progetto e copiare i file.



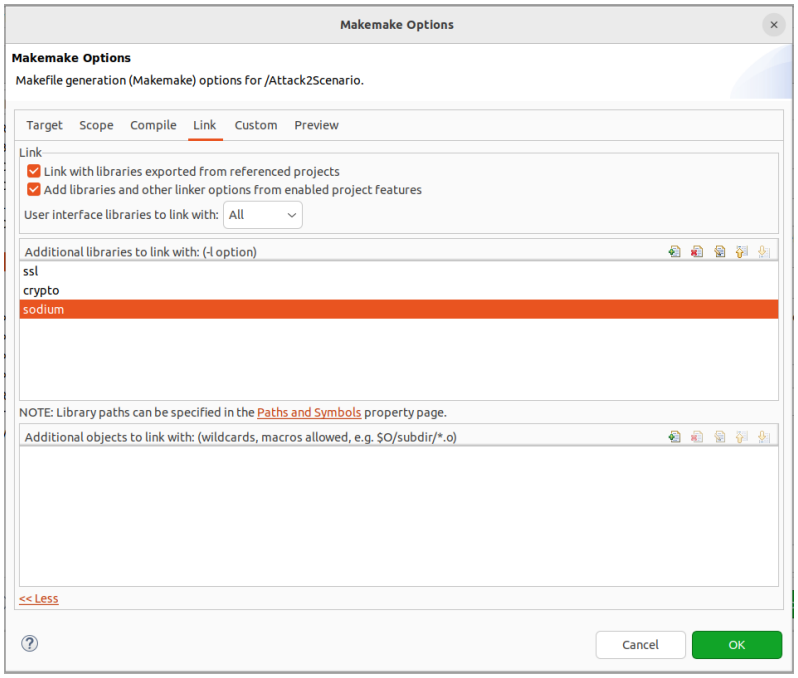
Per permettere al progetto di usare le classi di Inet 4.5, è necessario aggiungerlo alle *Project References*. Tasto destro sul Progetto → *Properties* → *Project References*, selezionare *inet4.5* e poi *Apply and Close*.

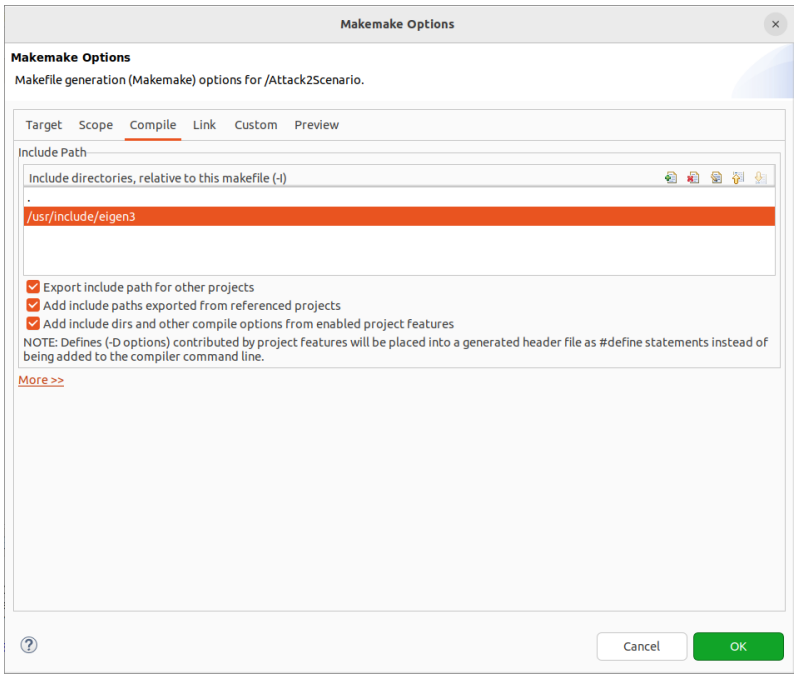


Inoltre per usare le librerie esterne, bisogna specificarle per la compilazione. Tasto destro sul Progetto → *Properties* → *Omnet++*, selezionare *Makemake* e poi selezionare il progetto in questione e cliccare su *Options*.



Si aprirà una finestra. OpenSSL e Sodium vanno "linkati", mentre Eigen va inclusa. Fare come nell'immagine che segue.





Il path `/usr/include/eigen3` è quello di default per Eigen. Nel caso in cui sia stata installata in un altro modo o con altri parametri, cercare ed inserire il nuovo path.

Nota L'aggiunta di tutte e tre le librerie potrebbe risultare inutile in tutti i progetti, infatti gli attacchi non usano la crittografia, né Eigen. Inoltre Eigen viene usata solo in una delle due mitigazioni del secondo scenario d'attacco.

2 Spiegazione del codice

Nella relazione finale del progetto è stato presentato, corredato di pseudocodice, non solo il meccanismo che permette di sfruttare GPSR ma anche le idee di attacco e mitigazione. Nella sezione che segue vengono presentate le modifiche attuate alla classe `Gpsr`, da INET 4.5, per ottenere la classe `GpsrStation` che implementa il protocollo GPSR con le estensioni.

2.1 Variabili introdotte

In `GpsrStation`

- **Parametri di configurazione**:
  - `n, m`: Dimensioni mappa e suddivisione in quadranti
  - `isMalicious`: Flag per nodi maliziosi
  - `isStation`: Flag per stazioni a terra
  - `sendMode`: Modalità di invio (`FixedMaximum/Interval`)
- **Dati statici**:
  - `timeInPerimeter/timeInGreedy`: Tempo accumulato nelle fasi di routing
  - `stationMap`: Mappa delle stazioni (indice → indirizzo)
  - `batteriesInfo`: Dati sul consumo energetico
  - `packetInfo`: Dati sul pacchetto
- **Timer aggiuntivi**:
  - `purgeStationTimer`: Pulizia delle posizioni ritenute obsolete dalle stazioni.
  - `sendTimerInterval`: Invio periodico di messaggi. Alla scadenza del timer:
    - Nel caso di `FixedMaximum` viene mandato un `selfMessage`, in base alla configurazione fornita nel file NED della rete, che ricorda al nodo di dover mandare un messaggio (il destinatario è esplicitato nel file NED della rete).
    - Nel caso di `Interval` viene mandato un `selfMessage` ogni t secondi, dove t è generato uniformemente da un intervallo [min, max] che può essere impostato nel file INI e ricorda al nodo di dover mandare un messaggio (il destinatario è scelto casualmente tra gli altri nodi).
  - `purgeDestinationTimer`: Pulizia delle destinazioni ritenute obsolete dai nodi.

2.2 Classi introdotte

`CustomStateBasedEpEnergyConsumer`

È la classe introdotta per la gestione dei consumi energetici. La classe è basata su `StateBasedEpEnergyConsumer` e si interfaccia con i moduli che gestiscono il comportamento del drone. La classe gestisce il consumo dettato dagli scambi dei messaggi in maniera del tutto analoga alla classe padre:

```
1  void CustomStateBasedEpEnergyConsumer::receiveSignal(cComponent *source,
       simsignal_t signal, intval_t value, cObject *details) {
2      Enter_Method("%s", cComponent::getSignalName(signal));
3      if (signal == physicallayer::IRadio::radioModeChangedSignal ||
4          signal == physicallayer::IRadio::receptionStateChangedSignal ||
5          signal == physicallayer::IRadio::transmissionStateChangedSignal ||
6          signal == physicallayer::IRadio::receivedSignalPartChangedSignal ||
7          signal == physicallayer::IRadio::transmittedSignalPartChangedSignal)
8      {
9          powerConsumption = computePowerConsumption();
10         emit(powerConsumptionChangedSignal, powerConsumption.get());
11     }
12     else{
```

```
        throw cRuntimeError("Unknown signal, RAD");
    }
}
```

In più, abbiamo introdotto dei metodi che gestiscono il consumo di energia della crittografia (quando si ricevono i segnali provenienti dal modulo `GpsrStation`, *signSignal* e *verifySignal*):

```
void CustomStateBasedEpEnergyConsumer::receiveSignal(cComponent *source,
    simsignal_t signal, double b, cObject *details){
    Enter_Method("%s", cComponent::getSignalName(signal));
    if(signal == inet::GpsrStation::signSignal){
        powerConsumption = computeSignConsumption(b);
        emit(powerConsumptionChangedSignal, powerConsumption.get());
    }
    else if(signal == inet::GpsrStation::verifySignal){
        powerConsumption = computeVerifyConsumption(b);
        emit(powerConsumptionChangedSignal, powerConsumption.get());
    }
    else{
        throw cRuntimeError("Unknown signal, CRYPT");
    }
}
```

```
W CustomStateBasedEpEnergyConsumer::computeSignConsumption(double bytes) const
{
    return W(bytes*signPowerConsumption);
}
```

```
W CustomStateBasedEpEnergyConsumer::computeVerifyConsumption(double bytes) const
{
    return W(bytes*verifyPowerConsumption);
}
```

Infine, i metodi che gestiscono il consumo relativo al movimento del drone, che si ottiene interfacciandosi al modulo di mobility.

```
void CustomStateBasedEpEnergyConsumer::receiveSignal(cComponent *source,
    simsignal_t signal, cObject *obj, cObject *details){
    Enter_Method("%s", cComponent::getSignalName(signal));
    if(signal == inet::IMobility::mobilityStateChangedSignal){
        powerConsumption = computeMovementConsumption();
        emit(powerConsumptionChangedSignal, powerConsumption.get());
    }
    else{
        throw cRuntimeError("Unknown signal, MOV");
    }
}
```

```
W CustomStateBasedEpEnergyConsumer::computeMovementConsumption() const
{
    double speed = mobility->getCurrentVelocity().length();
    return W(10*speed) + hoveringPowerConsumption;
}
```

2.3 Metodi introdotti

In `GpsrStation`

- **Gestione attacchi**:
  - `generateFalsePosition()`: Genera posizioni false
  - Nei `Beacon`, nell'invio della `StationNotice` o nelle risposte delle stazioni (in base all'attacco), uso condizionale: `if (isMalicious) setPosition(falsePosition)`
- **Comunicazione con le stazioni**:
  - `createStationNotice` (Nodo): crea una notifica di registrazione/deregistrazione

- `sendStationNotice` (Nodo → Stazione): manda una notifica di registrazione/deregistrazione alla stazione del proprio quadrante
- `processStationNotice` (Stazione): elabora una notifica di registrazione/deregistrazione
- `createPositionRequest` (Nodo): crea una richiesta di posizione
- `sendPositionRequest` (Nodo → Stazione): manda una richiesta di posizione ad una stazione
- `processPositionRequest` (Stazione): elabora una richiesta di posizione
- `createPositionResponse` (Stazione): crea una risposta contenente una posizione
- `sendPositionResponse` (Stazione → Nodo): manda una risposta contenente una posizione
- `processPositionResponse` (Nodo): elabora una risposta contenente una posizione
- `createS2SPositionRequest` (Stazione): crea una richiesta di posizione per un'altra stazione
- `sendS2SPositionRequest` (Stazione → Stazione): manda una richiesta di posizione ad un'altra stazione
- `processS2SPositionRequest` (Stazione): elabora una richiesta di posizione da un'altra stazione
- `createS2SPositionResponse` (Stazione): crea una risposta contenente una posizione per un'altra stazione
- `sendS2SPositionResponse` (Stazione → Stazione) : manda una risposta contenente una posizione ad un'altra stazione
- `processS2SPositionResponse` (Stazione): elabora una risposta contenente una posizione da un'altra stazione

- **Gestione Pacchetti**
  - `delayDatagram`: memorizza pacchetti in attesa di posizioni
  - `sendDelayedDatagram`: invia quando la posizione è disponibile
  - Modifiche a `datagramLocalOutHook` e `datagramPreRoutingHook`
- **Gestione posizioni**:
  - `getStationIndex`: calcola il quadrante di un nodo

$$q = \lfloor x/(n/m) \rfloor + m \cdot \lfloor y/(n/m) \rfloor$$

- **Statistiche**:
  - `finish`: Genera report JSON con:
    - Tempi di routing, messaggi inviati/ricevuti
    - Consumo batteria residuo
    - Pacchetti non consegnati

2.3.1 Modifiche a `datagramLocalOutHook` e `datagramPreRoutingHook`

`datagramLocalOutHook`

Scopo: Gestisce i pacchetti in uscita dal nodo, aggiungendo funzionalità per:

- Ritardare i pacchetti con destinazioni sconosciute
- Eliminare il filtro sui pacchetti di "SimpleMessage" in unicast
- Integrare le richieste alle stazioni

Codice Modificato

```
Result GpsrStation::datagramLocalOutHook(Packet *packet) {
    // ... controlli ...

    if (destination.isMulticast() || destination.isBroadcast()
    || routingTable->isLocalAddress(destination)||
            strstr(packet->getName(), simpleMessage) == nullptr)
        return ACCEPT;
    if (destinationsPositionTable.hasPosition(destination)) {
        // Crea opzione GPSR e inizia il routing
        GpsrOption *gpsrOption = createGpsrOption(destination);
        return routeDatagram(packet, gpsrOption);
    } else {
        // Metti in coda e richiedi posizione
        delayDatagram(packet);
        sendPositionRequest(destination, stationAddr);
        return QUEUE;
    }
}
```

`datagramPreRoutingHook`

Scopo: Intercetta i pacchetti in transito per:

- Prevenire eccezioni con controllo TTL

Codice Modificato

```
Result GpsrStation::datagramPreRoutingHook(Packet *datagram) {
    // ... controlli preliminari ...

    if (ipv4Header->getTimeToLive() <= 1) return DROP;

   // ... continua con le normali operazioni
}
```

2.4 Raccolta delle metriche e generazione del JSON

2.4.1 Struttura del JSON

Il file `output.json` contiene i seguenti campi:

- `PerimeterTime`: Tempo totale speso in modalità *perimeter routing*
- `GreedyTime`: Tempo totale in modalità *greedy routing*
- `totalMsg`: Numero totale di messaggi inviati
- `totalArrivedMsg`: Messaggi consegnati con successo
- `messages`: Dettagli per ogni messaggio consegnato
- `batteries`: Consumo energetico dei nodi
- `notArrivedMessages`: Messaggi non consegnati

2.4.2 Meccanismi di raccolta

Metriche di routing

- In `findGreedyRoutingNextHop` e `findPerimeterRoutingNextHop`:

```
auto start = std::chrono::high_resolution_clock::now();
// ... logica routing ...
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> duration = end - start;
timeInGreedy += duration.count(); // o timeInPerimeter

```

Tracking dei Pacchetti

- All'invio (`datagramLocalOutHook`) viene aggiornato il numero di pacchetti inviati e registrato l'istante di invio:

```
totalMsg++;
mapPacketSendTime[packetName] = simTime();

```

- Alla ricezione (`processMessage`) viene registrato l'istante di arrivo e il numero di hops del pacchetto, viene inoltre aggiornato il numero di messaggi arrivati:

```
packetInfo[msgName] = {simTime(), mapPacketHops[msgName]};
totalArrivedMsg++;

```

- All'inoltro (`routeDatagram`), se il risultato non è un DROP, viene aggiornato il numero di hops del pacchetto:

```
mapPacketHops[datagram->getFullName()]++;
```

Batteria

- All'inizializzazione (`handleStartOperation`):

```
batteriesInfo[hostname].first = getBatteryLevel(); // Livello iniziale

```

- Al termine (`finish`):

```
 for(auto& b: batteriesInfo){
     json info = {
          {"hostname", b.first},
          {"J consumed", b.second.first - b.second.second}
     };
      batteryArray.push_back(info);
  }
  data["batteries"] = batteryArray;
\end
```

2.4.3 Esempio di JSON Generato

```
{
  "PerimeterTime": 12.45,
  "GreedyTime": 8.32,
  "totalMsg": 150,
  "totalArrivedMsg": 142,
  "messages": [
    {
      "msgName": "SimpleMessage0",
      "time": "3.12s",
      "hops": 4,
      "sendTime": "1.05s"
    },
    // ... altri messaggi ...
  ],
  "batteries": [
    {
```

```
      "hostname": "node1",
      "J consumed": 105.3
    },
    // ... altri nodi ...
  ],
  "notArrivedMessages": [
    {
      "msgName": "SimpleMessage8",
      "hops": 3
    }
  ]
}
```

Note Importanti

- **Sincronizzazione**: L'aggiornamento delle metriche è thread-safe grazie alle strutture `static` (condivise tra tutte le istanze).
- **Performance**: L'uso di `std::chrono` garantisce alta precisione nella misurazione dei tempi.
- **Debug**: Il JSON include anche messaggi non consegnati, utili per analisi di affidabilità.

3 Particolari del codice dello Scenario d'attacco 1

L'attacco di falsificazione della posizione ai vicini si basa sulla manipolazione dei beacon, che contengono informazioni sulla posizione dei nodi. I nodi maliziosi generano e trasmettono beacon con posizioni false, ingannando gli altri nodi nella rete e compromettendo il corretto funzionamento del protocollo di routing e aumentando il tempo in perimeter mode.

3.1 Generazione di posizioni false

La funzione `generateFalsePosition()` è il cuore dell'attacco. Genera posizioni false utilizzando tre metodi diversi, selezionati in modo casuale:

- **Posizione Opposta nella Mappa:** La posizione falsa viene generata in un angolo opposto della mappa, con una coordinata Z elevata (100-120).
- **Posizione Opposta nel Quadrante:** La posizione falsa viene generata nel quadrante opposto rispetto alla posizione reale del nodo.
- **Posizione Casuale:** La posizione falsa viene generata in un punto casuale all'interno della mappa, con una coordinata Z elevata.

```
Coord GpsrStation::generateFalsePosition(){

    double prob = uniform(0.0, 1.0);
           if (prob <= 0.33){

               /* Opposta nella mappa */

               std::vector<Coord> angles;
               Coord ul = Coord();
               Coord ur = Coord();
               Coord dl = Coord();
               Coord dr = Coord();

               ul.setX(0);
               ul.setY(0);
               ul.setZ(0);

               ur.setX(n);
               ur.setY(0);
               ur.setZ(0);

               dl.setX(0);
               dl.setY(n);
               dl.setZ(0);

               dr.setX(n);
               dr.setY(n);
               dr.setZ(0);

               angles.push_back(ul);
               angles.push_back(ur);
               angles.push_back(dl);
               angles.push_back(dr);

               Coord pos = mobility->getCurrentPosition();

               double maxDistance = -1;
               int maxAngleIndex = 0;
               for (size_t i = 0; i < angles.size(); ++i) {
                   double currDist = pos.distance(angles[i]);
                   if (currDist > maxDistance){
                       maxDistance = currDist;
                       maxAngleIndex = i;
                   }
               }
```

```
46
47              Coord f = Coord();
48              Coord maxAngle = angles[maxAngleIndex];
49              double delta = n / m;
50              if (maxAngle == ul){
51                  /* UP LEFT */
52                  f.setX(uniform(0, delta));
53                  f.setY(uniform(0, delta));
54                  f.setZ(uniform(100, 120));
55              }
56              else if (maxAngle == ur){
57                  /* UP RIGHT */
58                  f.setX(uniform(n - delta, n));
59                  f.setY(uniform(0, delta));
60                  f.setZ(uniform(100, 120));
61              }
62              else if (maxAngle == dl){
63                  /* DOWN LEFT */
64                  f.setX(uniform(0, delta));
65                  f.setY(uniform(n - delta, n));
66                  f.setZ(uniform(100, 120));
67              }
68              else{
69                  /* DOWN RIGHT */
70                  f.setX(uniform(n - delta, n));
71                  f.setY(uniform(n - delta, n));
72                  f.setZ(uniform(100, 120));
73              }
74
75
76
77              return f;
78          }
79          else if (prob > 0.33 && prob <= 0.66){
80              /* Opposta nel quadrante */
81
82              Coord pos = mobility->getCurrentPosition();
83              double x = pos.getX();
84              double y = pos.getY();
85
86              double l = n / m;
87
88              int qX = std::floor(x / l);
89              int qY = std::floor(y / l);
90
91              if (qX >= m) qX = m - 1;
92              if (qY >= m) qY = m - 1;
93
94              double TL_x = qX * l;
95              double TL_y = qY * l;
96              double TR_x = (qX + 1) * l;
97              double TR_y = qY * l;
98              double BL_x = qX * l;
99              double BL_y = (qY + 1) * l;
100             double BR_x = (qX + 1) * l;
101             double BR_y = (qY + 1) * l;
102
103             double center_x = (TL_x + BR_x) / 2;
104             double center_y = (TL_y + BR_y) / 2;
105
106             double opposite_x = 2 * center_x - x;
107             double opposite_y = 2 * center_y - y;
108
109             Coord falsePosition = Coord();
110
111             falsePosition.setX(opposite_x);
112             falsePosition.setY(opposite_y);
113             falsePosition.setZ(uniform(100, 120));
114
115
```

```
116
117                 return falsePosition;
118             }
119             else{
120                 /* Random */
121
122                 Coord f = Coord();
123
124                 f.setX(uniform(0, n));
125                 f.setY(uniform(0, n));
126                 f.setZ(uniform(100, 120));
127
128
129
130
131                 return f;
132             }
133
134 }
```

Listing 1: Metodo generateFalsePosition

La funzione utilizza `uniform(0.0, 1.0)` per generare un numero casuale tra 0 e 1, che determina quale metodo di generazione della posizione falsa verrà utilizzato. In tutti i casi, la coordinata Z della posizione falsa viene impostata a un valore elevato, indicando potenzialmente un'altitudine irrealistica per un drone.

3.2 Creazione dei beacon modificati

La funzione `createBeacon()` crea un messaggio beacon contenente la posizione del nodo. Se il nodo è malizioso (`isMalicious` è vero), la posizione nel beacon viene sostituita con una posizione falsa generata da `generateFalsePosition()`.

```
1  const Ptr<GpsrBeacon> GpsrStation::createBeacon()
2  {
3      const auto& beacon = makeShared<GpsrBeacon>();
4          // MODIFICA
5      if (isMalicious){
6          beacon->setAddress(getSelfAddress());
7          beacon->setPosition(generateFalsePosition());
8          beacon->setChunkLength(B(getSelfAddress().getAddressType()->
       getAddressByteLength() + positionByteLength));
9      }
10     else{
11         beacon->setAddress(getSelfAddress());
12         beacon->setPosition(mobility->getCurrentPosition());
13         beacon->setChunkLength(B(getSelfAddress().getAddressType()->
       getAddressByteLength() + positionByteLength));
14     }
15     return beacon;
16 }
```

Listing 2: Metodo createBeacon

Se `isMalicious` è vero, la posizione nel beacon viene impostata sulla posizione falsa generata da `generateFalsePosition()`. Questo permette al nodo malizioso di inviare beacon con posizioni errate, ingannando gli altri nodi nella rete.

3.3 Elaborazione dei beacon

La funzione `processBeacon()` gestisce la ricezione dei beacon e aggiorna la tabella delle posizioni dei vicini.

```
1  void GpsrStation::processBeacon(Packet *packet)
2  {
3      const auto& beacon = packet->peekAtFront<GpsrBeacon>();
4      neighborPositionTable.setPosition(beacon->getAddress(), beacon->getPosition());
5      delete packet;
```

```
}
```

Listing 3: Metodo processBeacon

La funzione estrae la posizione e l'indirizzo del nodo dal beacon ricevuto e aggiorna la tabella `neighborPositionTable`. I nodi che ricevono beacon con posizioni false aggiorneranno la loro tabella dei vicini con informazioni errate, influenzando le decisioni di routing.

4 Particolari del codice della mitigazione per lo Scenario d'attacco 1

Descriviamo il meccanismo di mitigazione per contrastare l'attacco di falsificazione della posizione nel protocollo GPSR. Le funzioni analizzate includono l'elaborazione dei messaggi `StationNotice`, la creazione e l'elaborazione dei messaggi `StationNoticeResponse`, e la gestione dei beacon con firme digitali.

4.1 Comunicazione Nodo-Stazione

La funzione `processStationNotice()` gestisce i messaggi `StationNotice` ricevuti dalle stazioni.

```
void GpsrStation::processStationNotice(Packet *packet)
{
    packet->popAtFront<UdpHeader>();
    const auto& req = packet->peekAtFront<StationNotice>();
    L3Address address = req->getSource();
    Coord position = req->getPosition();
    if(req->getDeregister()){
        if(quadNodesPositionTable.hasPosition(address)){
            quadNodesPositionTable.removePosition(address);
        }
    }
    else{
        quadNodesPositionTable.setPosition(address, position);
        sendStationNoticeResponse(createStationNoticeResponse(position.getX(),
    position.getY(), position.getZ()),req->getSource(), req->getSourceModuleName())
    ;
    }
    delete packet;
}
```

Listing 4: Metodo processStationNotice

La funzione estrae l'indirizzo e la posizione del nodo dal messaggio `StationNotice`. Se il messaggio è di deregister, la posizione del nodo viene rimossa dalla tabella `quadNodesPositionTable`. Altrimenti, la posizione viene aggiunta alla tabella e viene inviato un messaggio di risposta `StationNoticeResponse` con la posizione certificata.

4.2 Il lavoro svolto dalla stazione

La funzione `createStationNoticeResponse()` crea un messaggio `StationNoticeResponse` contenente la posizione certificata del nodo.

```
const Ptr<StationNoticeResponse> GpsrStation::createStationNoticeResponse(double x,
     double y, double z){
     const auto& stationResponse = makeShared<StationNoticeResponse>();
     std::string clear = std::to_string(x) + "-" + std::to_string(y) + "-" + std::
    to_string(z);
     uint64_t nonce = nonceGen.generateNonce();
     std::string signature;
     if (signAlgorithm == ECDSA){
         signature=signMessageECDSA(keyPairECDSA, clear, nonce);
     }
     else{
         signature = signMessageEDDSA(clear, privateKeyEDDSA, nonce);
     }
     stationResponse->setC(signature.c_str());
     stationResponse->setX(x);
     stationResponse->setY(y);
     stationResponse->setZ(z);
     stationResponse->setNonce(nonce);
     stationResponse->setChunkLength(B(signature.length() + 16*3 + sizeof(nonce) +
    positionByteLength));
     return stationResponse;
```

```
}
```

Listing 5: Metodo createStationNoticeResponse

La funzione crea un messaggio di risposta `StationNoticeResponse` contenente le coordinate X, Y, Z della posizione del nodo, un nonce generato e una firma digitale calcolata utilizzando l'algoritmo di firma specificato (ECDSA o EDDSA).

La funzione `processStationNoticeResponse()` gestisce i messaggi `StationNoticeResponse` ricevuti.

```
void GpsrStation::processStationNoticeResponse(Packet *packet){
    packet->popAtFront<UdpHeader>();
    const auto& res = packet->peekAtFront<StationNoticeResponse>();
    std::string clear = std::to_string(res->getX()) + "-" + std::to_string(res->
    getY()) + "-" + std::to_string(res->getZ());
    Coord pos = Coord();
    pos.setX(res->getX());
    pos.setY(res->getY());
    pos.setZ(res->getZ());
    sendBeacon(createBeacon(pos, res->getC(), res->getNonce()));
    storeSelfPositionInGlobalRegistry();
    delete packet;
}
```

Listing 6: Metodo processStationNoticeResponse

La funzione estrae le coordinate X, Y, Z, la firma e il nonce dal messaggio `StationNoticeResponse`. Viene quindi creato e inviato un beacon utilizzando la posizione certificata e la firma.

4.3 Creazione dei Beacon con firma

La funzione `createBeacon()` crea un beacon contenente la posizione del nodo e la firma digitale.

```
const Ptr<GpsrBeacon> GpsrStation::createBeacon(Coord position, const char*
    signature, uint64_t nonce)
{
    const auto& beacon = makeShared<GpsrBeacon>();
        // MODIFICA
    if (isMalicious){
        beacon->setAddress(getSelfAddress());
        beacon->setPosition(generateFalsePosition());
        beacon->setSignature(signature);
        beacon->setNonce(nonce);
        beacon->setChunkLength(B(getSelfAddress().getAddressType()->
    getAddressByteLength() + positionByteLength));
    }
    else{
        beacon->setAddress(getSelfAddress());
        beacon->setPosition(position);
        beacon->setSignature(signature);
        beacon->setNonce(nonce);
        beacon->setChunkLength(B(getSelfAddress().getAddressType()->
    getAddressByteLength() + positionByteLength));
    }
    return beacon;
}
```

Listing 7: Metodo createBeacon (con firma)

La funzione crea un beacon con l'indirizzo, la posizione (falsa se il nodo è malizioso), la firma digitale e il nonce.

4.4 Gestione delle fiducia e verifica della firma nei beacon

La funzione `processBeacon()` gestisce la ricezione dei beacon e verifica la firma digitale.

```
void GpsrStation::processBeacon(Packet *packet)
{
    const auto& beacon = packet->peekAtFront<GpsrBeacon>();
```

```
    EV_INFO << "Processing beacon: address = " << beacon->getAddress() << ",
    position = " << beacon->getPosition() << endl;
    L3Address beaconAddress = beacon->getAddress();
    if(trustness[beaconAddress] <= notTrusted){
             int c=count(maliciousNode.begin(),maliciousNode.end(),beaconAddress);
                  if(c>0){
                          maliciousIndividuati.insert(beaconAddress);
                          //true negative
                          trueNegative += 1.0;
                   }
                   else{
                         falsiPositivi.insert(beaconAddress);
                         falseNegative += 1.0; //false negative
                   }
            delete packet;
     }
     else if(trustness[beaconAddress] >= trusted){
               if (count(maliciousNode.begin(), maliciousNode.end(), beaconAddress)
     > 0){
                         /* MALICIOUS NODE */
                         falsePositive += 1.0;
               }
               else{
                         /* GOOD NODE */
                         truePositive += 1.0;
               }
               neighborPositionTable.setPosition(beacon->getAddress(), beacon->
    getPosition());
               delete packet;
     }
     else{ //controllo
            Coord pos = beacon->getPosition();
            uint64_t nonce = beacon->getNonce();
            std::string clear = std::to_string(pos.getX()) + "-" + std::to_string(
    pos.getY()) + "-" + std::to_string(pos.getZ());
            int stationIndex = getStationIndex(pos);
            std::string signature = beacon->getSignature();
            bool verified=false;
            if (signAlgorithm == ECDSA){

                         verified = verifySignatureECDSA(keyPairStationMapECDSA[
    stationIndex],clear,nonce,signature);
                   }
                   else{
                         verified = verifySignatureEDDSA(clear, signature,
    publicKeyMapEDDSA[stationIndex], nonce);
                   }

            if(verified){

                double measurement = trustness[beaconAddress] + PENALTY;
                trustness[beaconAddress] = (1 - alpha) * trustness[beaconAddress] +
     alpha * measurement;
                neighborPositionTable.setPosition(beacon->getAddress(), beacon->
    getPosition());
            }
            else{
                double measurement = trustness[beaconAddress] - PENALTY;
                trustness[beaconAddress] = (1 - alpha) * trustness[beaconAddress] +
     alpha * measurement;
            }

            delete packet;
        }
}
```

Listing 8: Metodo processBeacon (con verifica firma)

La funzione `processBeacon()` gestisce la ricezione dei beacon e verifica la firma digitale per validare la posizione del nodo. La funzione opera in base al livello di fiducia del nodo mittente,

gestito dalla variabile `trustness`.

- **Nodi Non Fidati (`trustness[beaconAddress] <= notTrusted`):** Se il nodo mittente è considerato non fidato, la funzione verifica se l'indirizzo del nodo è presente nella lista dei nodi maliziosi (`maliciousNode`). Se presente, viene incrementato il contatore dei veri negativi (`trueNegative`); altrimenti, viene incrementato il contatore dei falsi negativi (`falseNegative`).
- **Nodi Fidati (`trustness[beaconAddress] >= trusted`):** Se il nodo mittente è considerato fidato, la funzione verifica se l'indirizzo del nodo è presente nella lista dei nodi maliziosi. Se presente, viene incrementato il contatore dei falsi positivi (`falsePositive`); altrimenti, viene incrementato il contatore dei veri positivi (`truePositive`). La posizione del nodo viene quindi aggiunta alla tabella `neighborPositionTable`.
- **Nodi con Fiducia Intermedia (Controllo):** Se il livello di fiducia del nodo è intermedio, la funzione verifica la firma digitale del beacon. La firma viene verificata utilizzando l'algoritmo specificato (ECDSA o EDDSA). Se la firma è valida, il livello di fiducia del nodo viene aumentato; altrimenti, viene diminuito. La posizione del nodo viene quindi aggiunta alla tabella `neighborPositionTable` solo se la firma è valida.

In ogni caso, il pacchetto viene eliminato dopo l'elaborazione.

5 Particolari del codice dello Scenario d'attacco 2

Lo Scenario d'attacco 2 estende la falsificazione di posizione a due livelli: nei beacon scambiati tra nodi e nelle comunicazioni dirette con le stazioni. Questo attacco mira a compromettere sia il routing locale che il sistema globale di registrazione delle posizioni.

Ecco l'analisi tecnica focalizzata sulle funzioni indicate:

5.1 Falsificazione posizione nei beacon

```
const Ptr<GpsrBeacon> GpsrStation::createBeacon()
{
    const auto& beacon = makeShared<GpsrBeacon>();
    beacon->setAddress(getSelfAddress());

    // Selezione strategia posizione
    if(isMalicious) {
        beacon->setPosition(generateFalsePosition()); // Generazione coordinata
    fittizia
    }
    else {
        beacon->setPosition(mobility->getCurrentPosition()); // Posizione reale
    }

    // Dimensionamento pacchetto
    beacon->setChunkLength(B(
        getSelfAddress().getAddressType()->getAddressByteLength() +
        positionByteLength // Parametro da file .ini
    ));

    return beacon;
}
```

Listing 9: Generazione beacon con posizione falsificata

Caratteristiche tecniche:

- Utilizzo di `makeShared` per gestione memoria ottimizzata
- Pattern `if-else` basato sul flag `isMalicious`
- Calcolo dimensione pacchetto tramite metadati dell'indirizzo (`addressByteLength`)
- Integrazione con modulo `mobility` per posizione reale

5.2 Falsificazione posizione nella StationNotice

```
const Ptr<StationNotice> GpsrStation::createStationNotice(bool deregisterFlag)
{
    const auto& stationNotice = makeShared<StationNotice>();
    stationNotice->setAddress(getSelfAddress());

    // Sovrascrittura posizione
    if(isMalicious){
        stationNotice->setPosition(generateFalsePosition());
    }
    else{
        stationNotice->setPosition(mobility->getCurrentPosition());
    }

    // Parametri aggiuntivi
    stationNotice->setDeregister(deregisterFlag);
    stationNotice->setChunkLength(B(
        getSelfAddress().getAddressType()->getAddressByteLength() +
        positionByteLength +
        1 // Flag deregistrazione
    ));

```

```
    return stationNotice;
}
```

Listing 10: Registrazione posizione malevola

Elementi implementativi:

- Gestione del flag `deregister` per invalidare registrazioni
- Dimensione pacchetto calcolata con componente fissa (1 byte per il flag)
- Riuso della logica di generazione posizioni false
- Utilizzo di `positionByteLength` configurabile

5.3 Generazione posizioni false

Funzione ausiliaria per entrambi i meccanismi:

```
Coord GpsrStation::generateFalsePosition(){
    double prob = uniform(0.0, 1.0);
    if (prob <= 0.33){
        // Strategia 1: Posizione opposta nella mappa
        // Calcoli geometrici per determinare angolo opposto
    }
    else if (prob > 0.33 && prob <= 0.66){
        // Strategia 2: Riflessione nel quadrante
        // Calcolo centroide e inversione coordinate
    }
    else{
        // Strategia 3: Posizione completamente casuale
        return Coord(
            uniform(0, n),
            uniform(0, n),
            uniform(minZ, maxZ)
        );
    }
}
```

Listing 11: Generazione coordinate fittizie

Implementazione multi-strategia:

- Distribuzione probabilistica delle tecniche di falsificazione
- Parametrizzazione dimensioni mappa (`n`) e granularità (`m`)
- Generazione deterministica di coordinate basate su geometria computazionale
- Pattern `if-else` per selezione strategia

6 Particolari del codice della mitigazione basata sulla triangolazione per lo Scenario d'attacco 2

6.1 Elaborazione di una StationNotice

```
void GpsrStation::processStationNotice(Packet *packet)
{
    // Rimozione header UDP e parsing del messaggio
    packet->popAtFront<UdpHeader>();
    const auto& req = packet->peekAtFront<StationNotice>();

    // Estrazione dati principali
    L3Address address = req->getAddress();
    Coord position = req->getPosition();
    const char* hostname = req->getSource();

    if(req->getDeregister()) { // Gestione deregistrazione
        quadNodesPositionTable.removePosition(address);
        addressNeighborsRSSMap.erase(address);
    } else { // Processo di validazione
        // Costruzione vettore di vicini con RSSI
        std::vector<std::tuple<L3Address, double, bool, simtime_t>>
    neighborsOfSender;
        for (size_t i = 0; i < req->getNeighborsArraySize(); i++) {
            std::tuple<L3Address, double, bool, simtime_t> pair(
                    req->getNeighbors(i),
                    req->getRss(i),
                    req->getTrusted(i),
                    req->getTimestamp(i));
            neighborsOfSender.push_back(pair);
        }

        // Aggiornamento strutture dati
        addressNeighborsRSSMap[address] = neighborsOfSender;
        quadNodesPositionTable.setPosition(address, position);

        // Raccolta dati per triangolazione
        std::vector<std::pair<Coord, double>> positionRSSvector;
        double trustCount = 0.0, trustTrue = 0.0;

        // Analisi cross-check con vicini
        for (const auto& [mainAddr, neighbors] : addressNeighborsRSSMap) {
            if(mainAddr == address) continue;

            for (const auto& [neighborAddr, rss, trusted, ts] : neighbors) {
                if(neighborAddr != address || !quadNodesPositionTable.hasPosition(
    mainAddr)) continue;

                // Conversione RSSI in distanza e validazione temporale
                if(simTime() - ts < positionValidityInterval) {
                    positionRSSvector.emplace_back(
                        quadNodesPositionTable.getPosition(mainAddr),
                        fromRSStoDistance(rss)
                    );

                    trustCount += 1.0;
                    if(trusted) trustTrue += 1.0;
                }
            }
        }

        // Calcolo posizione tramite trilaterazione
        double declaredDistance = -1;
        if(positionRSSvector.size() >= 4) {
            auto filtered = removeOutliers(positionRSSvector);
            if(filtered.size() >= 4) {
                Coord result = trilaterationGNWithMultiplePairsEx(filtered, ...);
                declaredDistance = position.distance(result);
            }
```

```
        } else if(positionRSSvector.size() == 3) {
            Coord result = simpleTrilateration2D(...);
            declaredDistance = position.distance(result);
        }

        // Aggiornamento trust dinamico
        double newTrust = 0.0;
        if(declaredDistance >= 0) {
            // Calcolo trust basato su distanza e vicini
            double distanceTrust = exp(-declaredDistance / distanceThreshold);
            double neighborTrust = trustTrue / trustCount;
            newTrust = alpha * newTrust + (1 - alpha) * previousTrust;

            // Applicazione filtro moving average
            filterMap[address].filter(result, simTime().dbl());
        } else {
            // Fallback con solo trust dei vicini
            newTrust = alpha * previousTrust + (1 - alpha) * (trustTrue /
    trustCount);
        }

        // Decisione finale e aggiornamento metriche
        trustness[address] = std::clamp(newTrust, Tmin, 1.0);
        double avgTrust = computeAverageTrust();

        if(trustness[address] < avgTrust) {
            // Nodo considerato malevolo
            quadNodesPositionTable.removePosition(address);
        } else {
            // Nodo considerato affidabile
            sendStationNoticeResponse(...);
        }
    }
    delete packet;
}
```

Listing 12: Validazione avanzata con RSSI e trust

Caratteristiche tecniche:

- **Strutture dati complesse**:
  - `std::map<L3Address, std::vector<std::tuple<...>>>` per memorizzare vicini e metriche RSSI
  - `std::vector<std::pair<Coord, double>>` per coppie posizione-distanza
- **Pattern di progettazione**:
  - Strategy per algoritmi di trilaterazione (2D vs Gauss-Newton)
  - Template Method per il calcolo del trust
- **Ottimizzazioni computazionali**:
  - Rimozione outliers con `removeOutliers()`
  - Filtro moving average (`filterMap`) per smoothing posizioni
- **Gestione memoria**:
  - Deallocazione esplicita con `delete packet`
- **Validazione temporale**:
  - Controllo `simTime() - ts < positionValidityInterval`
  - Timestamping con precisione a `simtime_t`

6.2 Triangolazione con Gauss-Newton

```
Coord GpsrStation::trilaterationGNWithMultiplePairsEx(
    const std::vector<std::pair<Coord, double>>& positionsWithDistances,
    double minX, double maxX, double minY, double maxY, double minZ, double maxZ)
{
    const int max_iterations = 200;
    const double tol = 1e-8;
    const double lambda_init = 1e-4;
    const double lambda_factor = 5.0;

    Coord pos = initialPositionEstimateForMultiplePairs(positionsWithDistances,
    minX, maxX, minY, maxY, minZ, maxZ);
    Eigen::VectorXd weights = calculateWeights(positionsWithDistances);
    Eigen::MatrixXd W = weights.asDiagonal();
    double lambda = lambda_init;
    double prev_error = std::numeric_limits<double>::max();

    for(int iter = 0; iter < max_iterations; ++iter) {
        Eigen::VectorXd residual = calculateResidualForMultiplePairs(pos,
    positionsWithDistances);
        Eigen::MatrixXd J = calculateJacobianForMultiplePairs(pos,
    positionsWithDistances);

        Eigen::MatrixXd WJ = W * J;
        Eigen::MatrixXd H = WJ.transpose() * J;
        Eigen::Vector3d g = WJ.transpose() * residual;

        H.diagonal() *= (1.0 + lambda);
        Eigen::Vector3d delta = H.colPivHouseholderQr().solve(-g);

        Coord new_pos(
            clamp(pos.x + delta.x(), minX, maxX),
            clamp(pos.y + delta.y(), minY, maxY),
            clamp(pos.z + delta.z(), minZ, maxZ)
        );

        double new_error = (W * calculateResidualForMultiplePairs(new_pos,
    positionsWithDistances)).norm();

        if(new_error < prev_error) {
            lambda /= lambda_factor;
            pos = new_pos;
            prev_error = new_error;
        } else {
            lambda *= lambda_factor;
        }

        if(std::abs(prev_error - new_error) < tol || delta.norm() < tol)
            break;
    }

    return pos;
}
```

Listing 13: Triangolazione con Gauss-Newton

Caratteristiche tecniche:

- **Ottimizzazione iterativa**:
  - Regolarizzazione adattiva con parametro λ
  - Criteri di convergenza multipli (variazione errore ¡ $1e^{-8}$ o norma delta ¡ $1e^{-8}$)
- **Algebra lineare**:
  - Utilizzo di Eigen per QR decomposition (`colPivHouseholderQr()`)
  - Matrice Jacobiana calcolata numericamente

- **Vincoli spaziali**:
  - Clamp delle coordinate entro `minX/Y/Z` e `maxX/Y/Z`
  - Supporto per coordinate 3D
- **Pesi dinamici**:
  - Calcolo basato sull'inverso delle distanze (`1.0/(distance + 1e-3)`)
  - Normalizzazione del vettore dei pesi

6.3 Triangolazione 2D

```
Coord GpsrStation::simpleTrilateration2D(
    Coord p1, double d1, Coord p2, double d2, Coord p3, double d3,
    double minX, double maxX, double minY, double maxY, double minZ, double maxZ)
{
    Eigen::Matrix2d A;
    Eigen::Vector2d B;

    A(0, 0) = 2 * (p2.x - p1.x);
    A(0, 1) = 2 * (p2.y - p1.y);
    A(1, 0) = 2 * (p3.x - p1.x);
    A(1, 1) = 2 * (p3.y - p1.y);

    B(0) = d1*d1 - d2*d2 - p1.x*p1.x + p2.x*p2.x - p1.y*p1.y + p2.y*p2.y;
    B(1) = d1*d1 - d3*d3 - p1.x*p1.x + p3.x*p3.x - p1.y*p1.y + p3.y*p3.y;

    Eigen::Vector2d X = A.colPivHouseholderQr().solve(B);

    return Coord(
        clamp(X(0), minX, maxX),
        clamp(X(1), minY, maxY),
        (minZ + maxZ)/2
    );
}
```

Listing 14: Triangolazione 2D

Caratteristiche tecniche:

- **Riduzione al sistema lineare**:
  - Equazioni derivate dalle differenze quadratiche delle distanze
  - Soluzione tramite QR decomposition
- **Assunzioni geometriche**:
  - Z calcolato come media tra `minZ` e `maxZ`
  - Utilizzato esclusivamente quando ci sono esattamente 3 punti validi
- **Gestione errori**:
  - Clamp delle coordinate entro i limiti spaziali

6.4 Elementi comuni ai metodi

- **Preprocessing**:
  - Rimozione outliers con `removeOutliers()` (basato su deviazione standard)
  - Filtro moving average (`filterMap`) per smoothing risultati
- **Strutture dati**:
  - `std::vector<std::pair<Coord, double>>` per coppie posizione-distanza

  - Eigen per operazioni matriciali (`Eigen::MatrixXd`, `Eigen::VectorXd`)

- **Parametrizzazione**:
  - Limiti spaziali (`minX/Y/Z`, `maxX/Y/Z`)
  - Costanti numeriche (`max_iterations=200`, `tol=1e-8`)

7 Particolari del codice della mitigazione basata sulle distanze per lo Scenario d'attacco 2

Il processo di mitigazione si articola attraverso diversi tipi di messaggi scambiati tra nodi e stazioni:

7.1 Comunicazione Nodo-Stazione

Quando un nodo entra nella rete o cambia la sua posizione, invia un messaggio `StationNotice` a una stazione per registrarsi. La stazione, dopo aver potenzialmente effettuato dei controlli (descritti nella sezione successiva), risponde con un messaggio `StationNoticeResponse` contenente la posizione del nodo mobile, firmata digitalmente dalla stazione stessa.

```
void GpsrStation::processStationNotice(Packet *packet)
{
    const auto& stationNotice = packet->peekAtFront<StationNotice>();
    L3Address stationAddress = stationNotice->getSenderAddress();
    Coord stationPosition = stationNotice->getPosition();
    bool deregister = stationNotice->getDeregister();

    EV_INFO << "Processing StationNotice from " << stationAddress << ", position =
    " << stationPosition << ", deregister = " << deregister << endl;

    if (!deregister) {
        if (trustness[stationAddress] > notTrusted2) {
            quadNodesPositionTable.setPosition(stationNotice->getNodeAddress(),
    stationNotice->getPosition());
            schedulePurgeNeighborsTimer();
        } else {
            EV_INFO << "Ignoring StationNotice from untrusted station " <<
    stationAddress << endl;
            quadNodesPositionTable.removePosition(stationNotice->getNodeAddress());
            if (containsKey(maliciousNode, stationNotice->getNodeAddress()))
                maliciousNode.erase(std::remove(maliciousNode.begin(),
    maliciousNode.end(), stationNotice->getNodeAddress()), maliciousNode.end());
            else
                notMaliciousNode.push_back(stationNotice->getNodeAddress());
        }
    } else {
        quadNodesPositionTable.removePosition(stationNotice->getNodeAddress());
        schedulePurgeNeighborsTimer();
    }
    delete packet;
}
```

Listing 15: Metodo processStationNotice

Quando un nodo mobile necessita della posizione di un altro nodo e non la possiede, può inviare un messaggio `PositionRequest` a una stazione fissa. La stazione, se conosce la posizione del nodo richiesto, risponde con un `PositionResponse`.Se la stazione iniziale a cui è stata inviata la `PositionRequest` non conosce la posizione del nodo richiesto, può inoltrare la richiesta ad altre stazioni utilizzando i messaggi `S2SPositionRequest` e `S2SPositionResponse`.

7.2 Controlli da Parte della Stazione

Le stazioni fisse implementano diversi controlli per validare le informazioni ricevute dai nodi:

7.2.1 Verifica della Firma nella StationNoticeResponse

Quando una stazione riceve un `StationNoticeResponse` da un nodo mobile, verifica la firma digitale inclusa nel messaggio. Questo assicura che la posizione riportata dal nodo sia stata effettivamente "certificata" dalla stazione stessa.

```
void GpsrStation::processStationNoticeResponse(Packet *packet)
{
    const auto& response = packet->peekAtFront<StationNoticeResponse>();
    L3Address nodeAddress = response->getAddress();
    Coord position = response->getPosition();
    uint64_t nonce = response->getNonce();
    std::string signature = response->getSignature();
    int stationIndex = getStationIndex(response->getStationPosition());
    bool verified = false;
    std::string clear = std::to_string(position.getX()) + "-" + std::to_string(
    position.getY()) + "-" + std::to_string(position.getZ());

    if (signAlgorithm == ECDSA) {
        verified = verifySignatureECDSA(keyPairStationMapECDSA[stationIndex], clear
    , nonce, signature);
    } else {
        verified = verifySignatureEDDSA(clear, signature, publicKeyMapEDDSA[
    stationIndex], nonce);
    }

    if (verified) {
        EV_INFO << "StationNoticeResponse verified for node " << nodeAddress << ",
    position = " << position << endl;
        // Invia un beacon firmato con la propria posizione
        sendBeacon(createBeacon(mobility->getCurrentPosition(), getSignature().
    c_str(), generateNonce()));
    } else {
        EV_WARN << "StationNoticeResponse verification failed for node " <<
    nodeAddress << endl;
        // Potrebbe essere necessario intraprendere ulteriori azioni, come ridurre
    la fiducia nel nodo
    }
    delete packet;
}
```

Listing 16: Metodo processStationNoticeResponse

7.2.2 Gestione della Fiducia

Le stazioni mantengono una valutazione della fiducia per ciascun nodo. Questa valutazione può essere influenzata da diversi fattori, come la coerenza delle informazioni di posizione riportate e il comportamento del nodo nella rete. Se la fiducia in un nodo scende al di sotto di una certa soglia, le informazioni di posizione fornite da quel nodo potrebbero essere ignorate.

7.3 Controlli da Parte dei Nodi

Anche i nodi implementano dei controlli per rilevare potenziali falsificazioni di posizione da parte dei loro vicini: Quando un nodo mobile riceve un beacon da un vicino, verifica la firma digitale inclusa nel beacon. Questa verifica viene effettuata utilizzando la chiave pubblica della stazione che ha registrato il vicino. Se la firma non è valida, il beacon viene considerato non attendibile.

```
void GpsrStation::processBeacon(Packet *packet)
{
    const auto& beacon = packet->peekAtFront<GpsrBeacon>();
    EV_INFO << "Processing beacon: address = " << beacon->getAddress() << ",
    position = " << beacon->getPosition() << endl;
    L3Address beaconAddress = beacon->getAddress();
    Coord position= beacon->getPosition();
    uint64_t nonce = beacon->getNonce();
    std::string signature = beacon->getSignature();

    std::string clear = std::to_string(position.getX()) + "-" + std::to_string(
    position.getY()) + "-" + std::to_string(position.getZ());
    int stationIndex = getStationIndex(position);
    bool verified=false;
    if (signAlgorithm == ECDSA){
        verified = verifySignatureECDSA(keyPairStationMapECDSA[stationIndex],clear,
    nonce,signature);
    }
    else{
        verified = verifySignatureEDDSA(clear, signature, publicKeyMapEDDSA[
    stationIndex], nonce);
    }

    if (verified){
        neighborPositionTable.setPosition(beaconAddress, position);
    }
    // ... (resto del metodo)
}
```

Listing 17: Metodo processBeacon (verifica firma)

7.3.1 Controllo di Consistenza della Distanza

Il nodo stima la distanza dal vicino basandosi sulla potenza del segnale ricevuto (RSSI) e la confronta con la distanza euclidea calcolata dalla posizione riportata nel beacon. Se la discrepanza tra le due distanze supera una certa soglia, il nodo mobile può considerare la posizione riportata dal vicino come sospetta e potenzialmente ridurre il suo livello di fiducia.

```
void GpsrStation::processBeacon(Packet *packet)
{
    // ... (parte iniziale del metodo)

    auto signal = packet -> getTag<inet::SignalPowerInd>();
    double rss;
    if(signal)
        rss = math::mW2dBmW(mW(signal->getPower()).get());
    double distance = fromRSStoDistance(rss);

    double realDistance = distanza(mobility->getCurrentPosition(),beacon->
    getPosition());
    double error= std::abs(realDistance - distance);

    if(isMalicious){
        mappaDistanzaIndirizzo[beaconAddress]=-1;
    }
    else if(error > 14){
        mappaDistanzaIndirizzo[beaconAddress]=-1; //valore fittio della distanza
    }else{
        mappaDistanzaIndirizzo[beaconAddress]=distance;
    }

    if(trustness[beaconAddress] <= notTrusted1){
```

```
24          delete packet;
25      }
26      else if(trustness[beaconAddress] >= trusted1){
27          neighborPositionTable.setPosition(beacon->getAddress(), beacon->getPosition
    ());
28          delete packet;
29      }
30      else{
31          delete packet;
32      }
33  }
```

Listing 18: Metodo processBeacon (controllo distanza)

7.3.2 Stima della Distanza Tramite RSSI

La funzione `fromRSStoDistance` implementa un modello di propagazione del segnale per stimare la distanza tra due nodi wireless basandosi sulla potenza del segnale ricevuto (RSSI). Questa stima è utilizzata nel protocollo per confrontare la distanza misurata con la distanza riportata nei beacon e rilevare potenziali anomalie.

Formula di Calcolo La funzione utilizza una formula derivata dal modello di propagazione a spazio libero (Free-Space Path Loss) con alcune modifiche. La formula implementata è la seguente:

$$d = 10^{\frac{P_t - P_r - C}{20}} \cdot \text{noiseFactor}$$

dove:

- d è la distanza stimata tra il trasmettitore e il ricevitore.
- P_t è la potenza del segnale trasmesso (in dBm).
- P_r è la potenza del segnale ricevuto (in dBm).
- C è una costante che dipende dalla frequenza del segnale.
- noiseFactor è un fattore di rumore casuale introdotto per simulare le variazioni ambientali.

La costante C è calcolata come:

$$C = 20 \log_{10}(f \cdot 10^9) + 20 \log_{10}\left(\frac{4\pi}{c}\right)$$

dove f è la frequenza del segnale in GHz e c è la velocità della luce (3×10^8 m/s).
Il codice C++ che implementa questa formula è il seguente:

```
1  double GpsrStation::fromRSStoDistance(double Pr){
2      double Pt= 3.01;
3      double frequency = 2.4;
4
5      double c = 3e8;
6      double C = 20*log10(frequency*1e9)+20*log10(4*M_PI/c);
7
8      double noiseFactor = 1.0 + (static_cast<double>(rand()) / RAND_MAX) * 0.1; /*
       dallo 0% al 10% in modo randomico*/
9
10     return pow(10,(Pt-Pr-C)/20.0)* noiseFactor;
11 }
```

Listing 19: Funzione fromRSStoDistance

Per gestire la comunicazione tra i nodi e le stazioni ed includere le informazioni, il messaggio di ‘stationNotice‘ viene modificato come segue:

```
class StationNotice extends FieldsChunk {
    L3Address source;
    string sourceModuleName;
    Coord position;
    bool deregister;
    L3Address addressList[];
    double distanceList[];
    Coord positionList[];
}
```

Listing 20: StationNotice

8 Particolari del codice dello Scenario d'attacco 3

L'attacco si basa sulla manipolazione delle informazioni sui droni ricevute dalle stazioni malevole. Inoltre, in presenza di mitigazione, l'attacco utilizzerà anche una funzione per tentare di generare una firma casuale. Vedremo questi metodi nelle sezioni successive.

8.1 Generazione di Posizioni False

La funzione `generateFalsePosition()` è il cuore dell'attacco. Genera posizioni false utilizzando tre metodi diversi, selezionati in modo casuale:

- **Posizione Opposta nella Mappa:** La posizione falsa viene generata in un angolo opposto alla stazione.
- **Posizione Opposta nel Quadrante:** La posizione falsa viene generata nel quadrante opposto rispetto alla posizione della stazione.
- **Posizione Casuale:** La posizione falsa viene generata in un punto casuale all'interno della mappa, con una coordinata Z elevata.

```
Coord GpsrStation::generateFalsePosition(){

    double prob = uniform(0.0, 1.0);
           if (prob <= 0.33){
               std::vector<Coord> angles;
               Coord ul = Coord();
               Coord ur = Coord();
               Coord dl = Coord();
               Coord dr = Coord();
               ul.setX(0);
               ul.setY(0);
               ul.setZ(0);
               ur.setX(n);
               ur.setY(0);
               ur.setZ(0);
               dl.setX(0);
               dl.setY(n);
               dl.setZ(0);
               dr.setX(n);
               dr.setY(n);
               dr.setZ(0);
               angles.push_back(ul);
               angles.push_back(ur);
               angles.push_back(dl);
               angles.push_back(dr);
               Coord pos = mobility->getCurrentPosition();
               double maxDistance = -1;
               int maxAngleIndex = 0;
               for (size_t i = 0; i < angles.size(); ++i) {
                   double currDist = pos.distance(angles[i]);
                   if (currDist > maxDistance){
                       maxDistance = currDist;
                       maxAngleIndex = i;
                   }
               }
               Coord f = Coord();
               Coord maxAngle = angles[maxAngleIndex];
               double delta = n / m;
               if (maxAngle == ul){
                   f.setX(uniform(0, delta));
                   f.setY(uniform(0, delta));
                   f.setZ(uniform(100, 120));
               }
               else if (maxAngle == ur){
                   f.setX(uniform(n - delta, n));
                   f.setY(uniform(0, delta));
                   f.setZ(uniform(100, 120));
```

```
                }
                else if (maxAngle == dl){
                    f.setX(uniform(0, delta));
                    f.setY(uniform(n - delta, n));
                    f.setZ(uniform(100, 120));
                }
                else{
                    f.setX(uniform(n - delta, n));
                    f.setY(uniform(n - delta, n));
                    f.setZ(uniform(100, 120));
                }
                return f;
            }
            else if (prob > 0.33 && prob <= 0.66){
                Coord pos = mobility->getCurrentPosition();
                double x = pos.getX();
                double y = pos.getY();
                double l = n / m;
                int qX = std::floor(x / l);
                int qY = std::floor(y / l);
                if (qX >= m) qX = m - 1;
                if (qY >= m) qY = m - 1;
                double TL_x = qX * l;
                double TL_y = qY * l;
                double TR_x = (qX + 1) * l;
                double TR_y = qY * l;
                double BL_x = qX * l;
                double BL_y = (qY + 1) * l;
                double BR_x = (qX + 1) * l;
                double BR_y = (qY + 1) * l;
                double center_x = (TL_x + BR_x) / 2;
                double center_y = (TL_y + BR_y) / 2;
                double opposite_x = 2 * center_x - x;
                double opposite_y = 2 * center_y - y;
                Coord falsePosition = Coord();
                falsePosition.setX(opposite_x);
                falsePosition.setY(opposite_y);
                falsePosition.setZ(uniform(100, 120));
                return falsePosition;
            }
            else{
                Coord f = Coord();
                f.setX(uniform(0, n));
                f.setY(uniform(0, n));
                f.setZ(uniform(100, 120));
                return f;
            }
}
```

Listing 21: Metodo generateFalsePosition

8.1.1 Analisi

La funzione utilizza `uniform(0.0, 1.0)` per generare un numero casuale tra 0 e 1, che determina quale metodo di generazione della posizione falsa verrà utilizzato. In tutti i casi, la coordinata Z della posizione falsa viene impostata a un valore elevato, indicando potenzialmente un'altitudine irrealistica per un drone.

8.2 Generazione della firma falsa

La funzione `createBeacon()` crea un messaggio beacon contenente la posizione del nodo. Se il nodo è malizioso (`isMalicious` è vero), la posizione nel beacon viene sostituita con una posizione falsa generata da `generateFalsePosition()`.

```
std::string GpsrStation::generateFakeSignatureEDDSA(std::string messageString,
    uint64_t nonce) {
    std::vector<unsigned char> message(messageString.begin(), messageString.end());
```

```
    std::vector<unsigned char> messageWithNonce(sizeof(nonce) + message.size());
    std::vector<unsigned char> fakeSignature(crypto_sign_BYTES + messageWithNonce.
    size());

    randombytes_buf(fakeSignature.data(), fakeSignature.size());
    return base64_encode(fakeSignature.data(), fakeSignature.size());
}
```

Listing 22: Metodo generateFakseSignatureEDDSA

8.2.1 Analisi

Se `isMalicious` è vero, la stazione tenta di generate una firma falsa casualmente. Questo perché avendo modificato la posizione, la firma precedente sarà sicuramente non valida. E' bene notare che il metodo ha solo valenza teorica perché tale generazione non porterà verosimilmente mai ad avere una firma consistente.

9 Particolari del codice della mitigazione per lo Scenario d'attacco 3

Il processo di mitigazione si articola attraverso diversi tipi di messaggi scambiati tra nodi mobili e stazioni fisse:

9.1 Comunicazione Nodo-Stazione

Quando un nodo entra nella rete o cambia la sua posizione, invia un messaggio `StationNotice` a una stazione fissa per registrarsi. La stazione, dopo aver verificato le firme digitali dei droni, memorizza le informazioni ricevute, firma e nonce compresi, in una struttura dati apposita, `quadNodesSignatureMap`.

```
void GpsrStation::processStationNotice(Packet *packet){
    packet->popAtFront<UdpHeader>();
    const auto& req = packet->peekAtFront<StationNotice>();
    L3Address address = req->getAddress();
    Coord position = req->getPosition();
    std::string droneSignature = req->getDroneSignature();
    uint64_t droneNonce = req->getDroneNonce();
    if(req->getDeregister()){
        if(quadNodesPositionTable.hasPosition(address)){
            quadNodesPositionTable.removePosition(address);
            if(quadNodesSignatureMap.find(address) != quadNodesSignatureMap.end()){
                quadNodesSignatureMap.erase(address);
            }
        }
    }
    else{
        std::string clear = std::to_string(position.getX()) + "-" + std::to_string(
    position.getY()) + "-" + std::to_string(position.getZ());
        bool verified = false;
        if (signAlgorithm == ECDSA){
            verified = verifySignatureECDSA(keyPairDroneMapECDSA[address], clear,
    droneNonce, droneSignature);
        }
        else{
            verified = verifySignatureEDDSA(clear, droneSignature,
    publicKeyDroneMapEDDSA[address], droneNonce);
        }
        if(verified) {
            quadNodesPositionTable.setPosition(address, position);
            std::tuple<Coord, std::string, uint64_t, simtime_t> tupla(position,
    droneSignature, droneNonce, simTime());
            quadNodesSignatureMap[address] = tupla;
        }
        else{
            if(quadNodesSignatureMap.find(address) != quadNodesSignatureMap.end()){
                quadNodesSignatureMap.erase(address);
            }
        }
    }
    delete packet;
}
```

Listing 23: Metodo processStationNotice

Quando un nodo mobile necessita della posizione di un altro nodo invia un messaggio `PositionRequest` a una stazione fissa. La stazione, se conosce la posizione del nodo richiesto, risponde con un `PositionResponse`. Se la stazione iniziale a cui è stata inviata la `PositionRequest` non conosce la posizione del nodo richiesto, può inoltrare la richiesta ad altre stazioni utilizzando i messaggi `S2SPositionRequest` e `S2SPositionResponse`; la risposta includerà la relativa signaure con relativo nonce recuperato da `quadNodesSignatureMap` e la firma (con nonce) generata dalla stazione stessa. Se la stazione è malevola, ignora tutti i controlli del caso, genera una posizione falsa, tenta di generare una firma falsa ed invia la risposta generata al nodo richiedente.

```
const Ptr<PositionResponse> GpsrStation::createPositionResponse(L3Address address)
{
    const auto& positionResponse = makeShared<PositionResponse>();
    positionResponse->setAddress(address);
    if(isMalicious) {
       Coord position = generateFalsePositionStation();
       positionResponse->setSetted(true);
       positionResponse->setPosition(position);
       positionResponse->setTime(simTime());
       std::string clear = std::to_string(position.getX()) + "-" +
                           std::to_string(position.getY()) + "-" +
                           std::to_string(position.getZ());
       uint64_t droneNonce = nonceGen.generateNonce();
       uint64_t stationNonce = nonceGen.generateNonce();
       positionResponse->setDroneSignature((generateFakeSignatureEDDSA(clear,
    droneNonce)).c_str());
       positionResponse->setStationSignature((generateFakeSignatureEDDSA(clear,
    stationNonce)).c_str());
       positionResponse->setDroneNonce(droneNonce);
       positionResponse->setStationNonce(stationNonce);
    }
    else{
        if (quadNodesPositionTable.hasPosition(address)){
            PositionTable::PositionWithTimestamp p = quadNodesPositionTable.
    getPositionWithTimestamp(address);
            std::tuple<Coord, std::string, uint64_t, simtime_t> tupla =
    quadNodesSignatureMap[address];
            Coord posInTuple = std::get<0>(tupla);
            if (posInTuple == p.position){
                positionResponse->setSetted(true);
                positionResponse->setPosition(p.position);
                positionResponse->setTime(p.timestamp);
                uint64_t droneNonce = std::get<2>(tupla);
                std::string droneSignature = std::get<1>(tupla);
                std::string clear = std::to_string(p.position.getX()) + "-" +
                                    std::to_string(p.position.getY()) + "-" +
                                    std::to_string(p.position.getZ());

                uint64_t stationNonce = nonceGen.generateNonce();
                std::string stationSignature;
                if (signAlgorithm == ECDSA){
                    stationSignature = signMessageECDSA(keyPairECDSA, clear,
    stationNonce);
                }
                else{
                    stationSignature = signMessageEDDSA(clear, privateKeyEDDSA,
    stationNonce);
                }
                positionResponse->setDroneSignature((droneSignature).c_str());
                positionResponse->setStationSignature((stationSignature).c_str());
                positionResponse->setDroneNonce(droneNonce);
                positionResponse->setStationNonce(stationNonce);
            }
            else{
                positionResponse->setSetted(false);
            }
        }
        else{
            positionResponse->setSetted(false);
        }
    }
    positionResponse->setChunkLength(B(1 + address.getAddressType()->
    getAddressByteLength()
            + positionByteLength
            + sizeof(positionResponse->getTime())
            + sizeof(positionResponse->getDroneSignature())
            + sizeof(positionResponse->getDroneNonce())
            + sizeof(positionResponse->getStationSignature())
            + sizeof(positionResponse->getStationNonce())
            ));
```

```
64      return positionResponse;
65  }
```

Listing 24: Metodo createPositionResponse

9.2 Controlli da Parte della Stazione

Le stazioni fisse implementano diversi controlli per validare le informazioni ricevute dai nodi mobili:

9.2.1 Verifica della Firma nella StationNoticeResponse

Quando una stazione riceve un `StationNoticeResponse` da un nodo mobile, verifica la firma digitale inclusa nel messaggio. Questo assicura che la posizione riportata dal nodo sia stata effettivamente "certificata" dalla stazione stessa.

```
void GpsrStation::processPositionResponse(Packet *packet){
    packet->popAtFront<UdpHeader>();
    const auto& res = packet->peekAtFront<PositionResponse>();
    bool setted = res->getSetted();
    L3Address address = res->getAddress();
    Coord position = res->getPosition();
    simtime_t time = res->getTime();
    std::string stationSignature = res->getStationSignature();
    uint64_t stationNonce = res->getStationNonce();
    std::string droneSignature = res->getDroneSignature();
    uint64_t droneNonce = res->getDroneNonce();
    if(setted){
        std::string clear = std::to_string(position.getX()) + "-" +
                            std::to_string(position.getY()) + "-" +
                            std::to_string(position.getZ());
        int stationIndex = getStationIndex(position);
        bool droneSignatureVerified = false;
        bool stationSignatureVerified = false;
        std::string senderStation = packet->getSenderModule()->getParentModule()->
    getFullName();

        if (signAlgorithm == ECDSA){
            droneSignatureVerified = verifySignatureECDSA(keyPairDroneMapECDSA[
    address], clear, droneNonce, droneSignature);
            stationSignatureVerified = verifySignatureECDSA(keyPairStationMapECDSA[
    stationIndex], clear, stationNonce, stationSignature);
        }
        else{
            droneSignatureVerified = verifySignatureEDDSA(clear, droneSignature,
    publicKeyDroneMapEDDSA[address], droneNonce);
            stationSignatureVerified = verifySignatureEDDSA(clear, stationSignature
    , publicKeyStationMapEDDSA[stationIndex], stationNonce);
        }

        if (droneSignatureVerified && stationSignatureVerified){
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                falsePositive++;
            else
                truePositive++;
            destinationsPositionTable.setPosition(address, position);
            if (hasDelayedDatagrams(address)){
                auto lt = targetAddressToDelayedPackets.lower_bound(address);
                auto ut = targetAddressToDelayedPackets.upper_bound(address);
                for (auto it = lt; it != ut; it++){
                    sendDelayedDatagram(it->second);
                }
                eraseDelayedDatagrams(address);
            }
        }
        else{
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                trueNegative++;
            else
                falseNegative++;
        }
    }
    delete packet;
```

```
}
```

Listing 25: Metodo processStationNoticeResponse

9.2.2 Verifica della Firma nella S2SPositionResponse

Quando una stazione riceve una `PositionResponse` verifica le due firme ricevute; se la verifica va a buon fine, provvederà ad inoltrare la posizione al drone richiedente, altrimenti ignora la risposta.

```
void GpsrStation::processS2SPositionResponse(Packet *packet){
    packet->popAtFront<UdpHeader>();
    const auto& res = packet->peekAtFront<S2SPositionResponse>();
    bool setted = res->getSetted();
    L3Address applicant = res->getApplicant();
    std::string applicantModuleName = res->getApplicantModuleName();
    L3Address address = res->getAddress();
    Coord position = res->getPosition();
    simtime_t time = res->getTime();
    std::string stationSignature = res->getStationSignature();
    uint64_t stationNonce = res->getStationNonce();
    std::string droneSignature = res->getDroneSignature();
    uint64_t droneNonce = res->getDroneNonce();
    if(setted){
        std::string clear = std::to_string(position.getX()) + "-" +
                            std::to_string(position.getY()) + "-" +
                            std::to_string(position.getZ());
        int stationIndex = getStationIndex(position);
        bool droneSignatureVerified = false;
        bool stationSignatureVerified = false;
        if (signAlgorithm == ECDSA){
            droneSignatureVerified = verifySignatureECDSA(keyPairDroneMapECDSA[
    address], clear, droneNonce, droneSignature);
            stationSignatureVerified = verifySignatureECDSA(keyPairStationMapECDSA[
    stationIndex], clear, stationNonce, stationSignature);
        }
        else{
            droneSignatureVerified = verifySignatureEDDSA(clear, droneSignature,
    publicKeyDroneMapEDDSA[address], droneNonce);
            stationSignatureVerified = verifySignatureEDDSA(clear, stationSignature
    , publicKeyStationMapEDDSA[stationIndex], stationNonce);
        }

        std::string senderStation = packet->getSenderModule()->getParentModule()->
    getFullName();
        if (droneSignatureVerified && stationSignatureVerified) {
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                falsePositive++;
            else
                truePositive++;
        }
        else {
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                trueNegative++;
            else
                falseNegative++;
        }
        if ((droneSignatureVerified && stationSignatureVerified) || isMalicious){
            sendPositionResponse(createPositionResponseFromAnotherStation(address,
    position, time, droneSignature, droneNonce, stationSignature, stationNonce),
                    applicant, applicantModuleName.c_str());
        }
    }
    delete packet;
}
```

Listing 26: Metodo processS2SPositionResponse

9.3 Controlli da Parte dei Nodi

Anche i nodi mobili implementano dei controlli per rilevare potenziali falsificazioni di posizione da parte delle stazioni; anche in questo caso il meccanismo di mitigazione fa affidamento alla verifica della doppia firma, quella del drone destinazione e quella della stazione che ha inviato la risposta.

```
void GpsrStation::processPositionResponse(Packet *packet){
    packet->popAtFront<UdpHeader>();
    const auto& res = packet->peekAtFront<PositionResponse>();
    bool setted = res->getSetted();
    L3Address address = res->getAddress();
    Coord position = res->getPosition();
    simtime_t time = res->getTime();
    std::string stationSignature = res->getStationSignature();
    uint64_t stationNonce = res->getStationNonce();
    std::string droneSignature = res->getDroneSignature();
    uint64_t droneNonce = res->getDroneNonce();
    if(setted){
        std::string clear = std::to_string(position.getX()) + "-" +
                            std::to_string(position.getY()) + "-" +
                            std::to_string(position.getZ());
        int stationIndex = getStationIndex(position);
        bool droneSignatureVerified = false;
        bool stationSignatureVerified = false;
        std::string senderStation = packet->getSenderModule()->getParentModule()->
    getFullName();
        if (signAlgorithm == ECDSA){
            droneSignatureVerified = verifySignatureECDSA(keyPairDroneMapECDSA[
    address], clear, droneNonce, droneSignature);
            stationSignatureVerified = verifySignatureECDSA(keyPairStationMapECDSA[
    stationIndex], clear, stationNonce, stationSignature);
        }
        else{
            droneSignatureVerified = verifySignatureEDDSA(clear, droneSignature,
    publicKeyDroneMapEDDSA[address], droneNonce);
            stationSignatureVerified = verifySignatureEDDSA(clear, stationSignature
    , publicKeyStationMapEDDSA[stationIndex], stationNonce);
        }
        if (droneSignatureVerified && stationSignatureVerified){
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                falsePositive++;
            else
                truePositive++;
            destinationsPositionTable.setPosition(address, position);
            if (hasDelayedDatagrams(address)){
                auto lt = targetAddressToDelayedPackets.lower_bound(address);
                auto ut = targetAddressToDelayedPackets.upper_bound(address);
                for (auto it = lt; it != ut; it++){
                    sendDelayedDatagram(it->second);
                }
                eraseDelayedDatagrams(address);
            }
        }
        else{
            if(count(maliciousStations.begin(), maliciousStations.end(),
    senderStation) > 0)
                trueNegative++;
            else
                falseNegative++;
        }
    }
    delete packet;
}
```

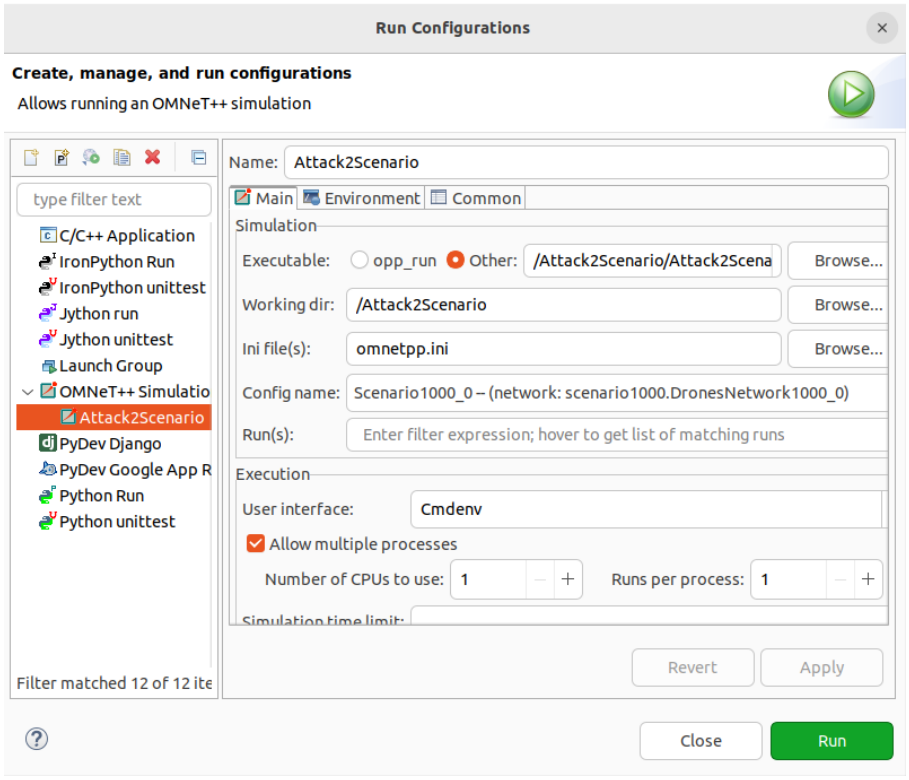
Listing 27: Metodo processS2SPositionResponse

10 Istruzioni e comandi per lanciare le simulazioni

10.1 Tramite Omnet++

Cliccando col tasto sul progetto e poi *Run Configurations*, dopo aver scelto lo scenario, si può scegliere come lanciare la simulazione:

- *Qtenv* per lanciarla in ambiente "grafico", sia in 2D che in 3D
- *Cmdenv* per lanciarla solo da riga di comando



10.2 Script personalizzati

Il lancio delle simulazioni si basa sull'esecuzione di due script, `generate_scenario.py` e `launch_simulations.sh`. Il primo script è utilizzato per generare i file *omnetpp.ini* e i relativi file ned. La sintassi per lanciare correttamente il comando (relativo alla mitigazione) è la seguente:

```
./generate_scenario.py --droneGridSize <radice_num_droni> \
--areaMin <dim_area_minima> \
--areaMax <dim_area_massima> --malMin <num_min_nodi_malevoli> \
--malMax <num_max_nodi_malevoli> --nQuad <num_quadranti> \
--nMsg <num_msg_generati> --simTime <tempo_simulazione> \
--minInterval <intervallo_minimo_di_generazione_msg> \
--maxInterval <intervallo_massimo_di_generazione_msg> \
--sendMode <modalità_di_invio> --signAlgorithm <algoritmo_firma_dig>
```

dove:

- `sendMode` è *"FixedMaximum"* o *"Interval"*
- `signAlgorithm` è assente nel file relativo all'attacco, mentre in quello relativo alla mitigazione è *"ECDSA"* o *"EDDSA"*.

- `malMin` e `malMax` sono dei valori compresi tra 0 e 40; nello scenario 1 e 2 si riferiscono al numero di droni, nello scenario 3 si riferiscono alla percentuale di stazioni malevole, arrotondato per eccesso.

Il secondo script è quello utilizzato per avviare la simulazione vera e propria. La sintassi è simile a quella dello script precedente:

```
./launch_simulations.sh --areaMin <dim_area_minima> \
--areaMax <dim_area_massima> --malMin <num_min_nodi_malevoli> \
--malMax <num_max_nodi_malevoli> --nQuad <num_quadranti> \
--nSim <numero_di_simulazioni> --gridSize <radice_num_droni> \
--nMsg <num_msg_generati> --simTime <tempo_simulazione> \
--minInterval <intervallo_minimo_di_generazione_msg> \
--maxInterval <intervallo_massimo_di_generazione_msg> \
--sendMode <modalità_di_invio> --signAlgorithm <algoritmo_firma_dig>
```

dove:

- `nSim` indica il numero di simulazioni da lanciare per scenario considerato. Per esempio, se si considera uno scenario con dimensione minima equivalente alla dimensione massima (*–areaMin* 1000 e *––areaMax* 1000) e con malevoli varianti da 0 a 20 (*–malMin* 0 e *–malMax 20*), `launch_simulations.sh` lancerà un numero di simulazioni pari ad *-nSim* per ogni file ned generato da `generate_scenario.py`, che nel nostro caso sono *DronesNetwork1000_0.ned*, *Drones1000_10.ned*, *DronesNetwork1000_20.ned*.

Lo script `launch_simulations.sh` lancia contestualmente `generate_scenario.py`, dunque non è necessario avviare lo script per la generazione degli scenari. I risultati delle varie simulazioni sono salvati nella cartella */results/json* della simulazione avviata. Nel caso in cui siano state fatte modifiche al codice, occorre rebuildare l'eseguibile da omnet prima di eseguire lo script.

Nota Per salvare correttamente i file JSON è necessario che nella directory del progetto esista una sottocartella `json` nella cartella `results`, ossia `results/json`.