

# GPSR modificato con stazioni a terra: un esempio di tre attacchi e relative mitigazioni

Aloia, Arena, Nasso, Saporito

# Contents

<b>1 Gpsr</b>	<b>5</b>
1.0.1 Introduzione . . . . .	5
1.0.2 La planarizzazione del Grafo . . . . .	5
1.0.3 Combinazione delle Modalità di Routing . . . . .	6
<b>2 Gpsr con stazioni a terra: scenario ed architettura generale</b>	<b>9</b>
2.1 Messaggi . . . . .	9
2.2 Strutture dati . . . . .	9
2.3 Registrazione e deregistrazione dei nodi . . . . .	11
2.4 Richiesta ed ottenimento della posizione di un nodo . . . . .	13
<b>3 Scenario d'attacco 1: falsificazione della posizione agli altri nodi</b>	<b>15</b>
3.1 Introduzione . . . . .	15
3.2 Attacco . . . . .	15
3.3 Implementazione . . . . .	17
3.4 Valutazione delle performance . . . . .	18
3.4.1 60 secondi . . . . .	19
3.4.2 120 secondi . . . . .	20
<b>4 L'utilizzo della crittografia</b>	<b>21</b>
4.1 Introduzione . . . . .	21
4.2 Crittografia asimmetrica e curve ellittiche . . . . .	21
<b>5 Mitigazione per falsificazione della posizione agli altri nodi (Scenario d'attacco 1)</b>	<b>23</b>
5.1 Introduzione . . . . .	23
5.2 Meccanismo di mitigazione . . . . .	23
5.3 Elaborazione dei dati da parte della stazione . . . . .	25
5.4 Dettagli dell'algoritmo <i>processStationNotice</i> . . . . .	25
5.5 Dettagli dell'algoritmo <i>createStationNoticeResponse</i> . . . . .	26
5.6 Dettagli dell'algoritmo <i>createBeacon</i> . . . . .	28
5.7 Meccanismo di trustness . . . . .	30
5.8 Calcolo del valore di trustness . . . . .	30
5.9 Lo smoothing . . . . .	31
5.10 Algoritmo di elaborazione dei beacon . . . . .	31
5.11 Implementazione . . . . .	32
5.12 Valutazione delle performance . . . . .	34
5.12.1 60 secondi . . . . .	35
5.12.2 120 secondi . . . . .	38
5.12.3 Confronto fra ECDSA e EDDSA 60 secondi . . . . .	40
5.12.4 Confronto fra ECDSA e EDDSA 120 secondi . . . . .	42
5.13 Conclusioni . . . . .	44

<b>6 Scenario d'attacco 2: falsificazione della posizione agli altri nodi ed alle stazioni</b>	<b>45</b>
6.1 Introduzione . . . . .	45
6.2 Attacco . . . . .	45
6.3 Conseguenze . . . . .	46
6.4 Implementazione . . . . .	47
6.5 Valutazioni delle performance . . . . .	48
6.5.1 60 secondi . . . . .	49
6.5.2 120 secondi . . . . .	50
<b>7 Mitigazione per falsificazione della posizione agli altri nodi ed alle stazioni con triangolazione (Scenario d'attacco 2)</b>	<b>51</b>
7.1 Meccanismo di mitigazione . . . . .	51
7.2 Elaborazione di una StationNotice da parte delle stazioni . . . . .	53
7.3 Dettagli dell'Algoritmo . . . . .	53
7.4 Comportamento dei nodi malevoli . . . . .	56
7.5 Stima della distanza da RSS . . . . .	56
7.6 Calcolo del valore di <i>trustness</i> . . . . .	56
7.6.1 Parametri e variabili usate . . . . .	57
7.6.2 Calcolo . . . . .	57
7.6.3 Calcolo della soglia di <i>trustness</i> minima . . . . .	58
7.7 Triangolazione . . . . .	59
7.7.1 Algoritmo con 3 ancore . . . . .	59
7.7.2 Algoritmo con $N \geq 4$ ancore . . . . .	60
7.7.3 Tecniche di stabilizzazione dei risultati . . . . .	60
7.8 Valutazione delle performance . . . . .	62
7.8.1 60 secondi . . . . .	63
7.8.2 120 secondi . . . . .	66
7.8.3 Confronto fra ECDSA e EDDSA 60 secondi . . . . .	68
7.8.4 Confronto fra ECDSA e EDDSA 120 secondi . . . . .	70
7.9 Conclusioni . . . . .	71
<b>8 Mitigazione per falsificazione della posizione agli altri nodi ed alle stazioni senza triangolazione (Scenario d'attacco 2)</b>	<b>72</b>
8.1 Definizioni e notazioni . . . . .	72
8.2 Aggiornamento della trustness . . . . .	73
8.2.1 Stima della distanza . . . . .	73
8.3 Verifica basata sulle distanze . . . . .	74
8.4 Gestione dei nodi maliziosi . . . . .	75
8.5 Valutazione delle performance . . . . .	76
8.5.1 60s . . . . .	77
8.5.2 120s . . . . .	80
8.5.3 Confronto fra ECDSA e EDDSA 60 secondi . . . . .	82
8.5.4 Confronto fra ECDSA e EDDSA 120 secondi . . . . .	84
8.6 Conclusioni . . . . .	86

<b>9 Scenario d'attacco 3: stazioni inaffidabili o compromesse</b>	<b>87</b>
9.1 Introduzione . . . . .	87
9.2 Logica dell'attacco . . . . .	88
9.3 Generazione di posizioni false . . . . .	90
9.4 Valutazioni delle performance . . . . .	91
9.4.1 60 secondi . . . . .	92
9.4.2 120 secondi . . . . .	93
9.5 Conseguenze . . . . .	93
<b>10 Mitigazione per stazioni inaffidabili o compromesse (Scenario d'attacco 3)</b>	<b>94</b>
10.1 Identificazione . . . . .	94
10.2 Diagramma di sequenza della mitigazione . . . . .	95
10.3 Limitazioni dei danni . . . . .	98
10.4 Valutazione delle performance . . . . .	102
10.4.1 60 secondi . . . . .	103
10.4.2 120 secondi . . . . .	106
10.4.3 Confronto fra ECDSA e EDDSA 60 secondi . . . . .	108
10.4.4 Confronto fra ECDSA e EDDSA 120 secondi . . . . .	110
10.5 Conclusioni . . . . .	111

# 1 Gpsr

## 1.0.1 Introduzione

Il protocollo GPSR [1] (Greedy Perimeter Stateless Routing) è stato concepito per le reti wireless in cui i nodi conoscono le proprie coordinate geografiche. La particolarità di GPSR risiede nella sua natura *stateless*: ogni nodo mantiene informazioni solo sui propri vicini immediati, senza dover memorizzare l'intera topologia della rete. Questa caratteristica permette al protocollo di scalare efficacemente anche in ambienti altamente dinamici, dove la mobilità e il numero di nodi possono essere elevati.

**Greedy Forwarding** Nella modalità **greedy forwarding**, un nodo inoltra il pacchetto al vicino che risulta geograficamente più vicino alla destinazione specificata nel pacchetto. Questo approccio, basato esclusivamente sulle informazioni locali, permette di costruire un percorso che avvicina progressivamente il pacchetto al target. Tuttavia, in alcune topologie, un nodo può trovarsi in un *local maximum* (massimo locale), cioè in una situazione in cui nessuno dei vicini è più vicino alla destinazione rispetto al nodo stesso.

**Perimeter Forwarding** Quando la forwardazione greedy fallisce a causa di un local maximum, il protocollo passa alla modalità **perimeter forwarding**. In questa modalità, il pacchetto viene instradato lungo il perimetro di una regione priva di vicini utili, utilizzando la *right-hand rule* (regola della mano destra). In pratica, il nodo inoltra il pacchetto seguendo l'ordine antiorario dei suoi vicini, in modo da far circumnavigare il *void* (zona priva di nodi più vicini alla destinazione) e riprendere successivamente la modalità greedy quando il pacchetto raggiunge una posizione più favorevole.

## 1.0.2 La planarizzazione del Grafo

La planarizzazione è un passaggio fondamentale per il corretto funzionamento della modalità di perimeter forwarding in GPSR. In un grafo planare, infatti, nessun paio di archi si interseca, consentendo così l'applicazione affidabile della regola della mano destra per instradare i pacchetti lungo i perimetri delle regioni prive di vicini utili (i cosiddetti *void*). Per ottenere un grafo planare, vengono adottati metodi che eliminano gli archi in eccesso, preservando però la connettività della rete. Tra i metodi più diffusi nel contesto di GPSR troviamo il **Relative Neighborhood Graph (RNG)** e il **Gabriel Graph (GG)**.

**Relative Neighborhood Graph (RNG)** Nel RNG, un arco  $(u, v)$  è incluso se e solo se non esiste alcun nodo  $w$  (diverso da  $u$  e  $v$ ) tale che la distanza tra  $u$  e  $v$  sia maggiore della massima tra le distanze  $d(u, w)$  e  $d(v, w)$ . Formalmente:

$$(u, v) \in RNG \iff \nexists w \neq u, v : d(u, v) > \max\{d(u, w), d(v, w)\}. \quad (1)$$

Questo criterio assicura che l'arco  $(u, v)$  venga mantenuto solo se l'area geometrica definita (il *lune* formato dall'intersezione dei due cerchi centrati in  $u$  e  $v$  con raggio  $d(u, v)$ ) è priva di altri nodi. Il risultato è un grafo più sparso, in cui molti archi ridondanti vengono eliminati, facilitando l'applicazione della regola della mano destra durante la fase di perimeter forwarding.

**Gabriel Graph (GG)** Il Gabriel Graph utilizza un criterio differente: un arco  $(u, v)$  è incluso nel grafo se e solo se il cerchio avente come diametro il segmento che collega  $u$  e  $v$  non contiene alcun altro nodo. In termini matematici:

$$(u, v) \in GG \iff \nexists w \neq u, v : d^2(u, v) < d^2(u, w) + d^2(v, w). \quad (2)$$

Questo vincolo implica che l'arco  $(u, v)$  sia mantenuto solo se il cerchio definito ha un'area completamente libera da altri nodi. Il GG tende a produrre un grafo con una connettività leggermente maggiore rispetto al RNG, in quanto il criterio di inclusione degli archi è meno restrittivo.

**Confronto e Considerazioni** Entrambi i metodi garantiscono la planarità del grafo, requisito indispensabile per l'applicazione efficace del perimeter forwarding in GPSR. Tuttavia, presentano differenze significative:

- **RNG:** Elimina un maggior numero di archi, producendo un grafo più sparso. Ciò può ridurre il traffico di controllo e semplificare il processo di routing, ma a volte potrebbe ridurre la ridondanza delle rotte.
- **GG:** Mantiene un numero maggiore di collegamenti, offrendo una maggiore diversità nelle possibili rotte, anche se ciò può tradursi in un traffico leggermente superiore.

La scelta tra RNG e GG dipende dal compromesso desiderato tra efficienza (minimizzazione del traffico di controllo) e robustezza (maggiore diversità delle rotte) nel contesto del routing.

Questi algoritmi di planarizzazione consentono a GPSR di mantenere la scalabilità, poiché ogni nodo necessita di conoscere solamente le posizioni dei vicini immediati. Inoltre, permettono di applicare la regola della mano destra per instradare i pacchetti intorno ai void, garantendo così un'alta probabilità di successo nel raggiungere la destinazione anche in presenza di topologie complesse o dinamiche.

### 1.0.3 Combinazione delle Modalità di Routing

GPSR integra due modalità di instradamento distinte, ognuna delle quali è ottimizzata per condizioni topologiche specifiche della rete. Inizialmente, il protocollo adotta la modalità di *greedy forwarding*, in cui un nodo inoltra il pacchetto al vicino che risulta geograficamente più vicino alla destinazione. Questa strategia, basata esclusivamente sulle informazioni locali, consente un rapido avvicinamento verso il target. Tuttavia, in presenza di un *local maximum* – ovvero quando il nodo corrente non dispone di nessun vicino che riduca la distanza al destinatario – la modalità greedy diventa inefficace.

In tali casi, GPSR passa alla modalità di *perimeter forwarding*, sfruttando la planarizzazione del grafo. Il protocollo utilizza la regola della mano destra per instradare il pacchetto lungo il perimetro di una regione priva di vicini utili (il cosiddetto *void*). Il processo di transizione e integrazione avviene secondo i seguenti passi:

- **Rilevamento del Local Maximum:** Quando il nodo corrente non identifica nessun vicino più vicino alla destinazione, viene rilevato un local maximum. In questo momento, il nodo attiva la modalità di perimeter forwarding.
- **Memorizzazione del Punto di Transizione:** Al passaggio in modalità perimeter, il nodo registra la posizione  $L_p$ , ovvero il punto in cui la modalità greedy ha fallito. Questo valore funge da riferimento per valutare, in seguito, se è possibile ritornare alla modalità greedy.

- **Applicazione della Right-Hand Rule:** In modalità perimeter, il pacchetto viene instradato seguendo la regola della mano destra. Il pacchetto percorre il perimetro del void, ovvero gli spigoli del grafo planare, fino a raggiungere un punto in cui si riduce la distanza alla destinazione.
- **Ritorno alla Modalità Greedy:** Se durante il percorso il pacchetto raggiunge un nodo il cui distacco dalla destinazione è inferiore rispetto al punto  $L_p$ , il protocollo ripristina la modalità greedy per proseguire l'inoltro in maniera diretta.

Questa combinazione sinergica permette di:

- **Minimizzare lo stato locale:** Ogni nodo mantiene informazioni solamente sui propri vicini immediati, rendendo il protocollo altamente scalabile.
- **Gestire in maniera dinamica i local maximum:** Il passaggio a perimeter forwarding consente di recuperare il percorso anche quando la modalità greedy fallisce.
- **Ottimizzare il percorso di instradamento:** Il ritorno alla modalità greedy, quando possibile, assicura che il pacchetto prosegua il percorso più diretto verso la destinazione.
- **Adattarsi in tempo reale:** La capacità di passare agevolmente da una modalità all'altra consente al protocollo di adattarsi rapidamente alle variazioni della topologia della rete, garantendo un instradamento robusto anche in ambienti ad alta mobilità.

In sintesi, l'integrazione delle modalità greedy e perimeter forwarding in GPSR permette di ottenere un bilanciamento ottimale tra efficienza e robustezza, assicurando che il percorso del pacchetto venga adattato dinamicamente in base alle condizioni locali della rete.

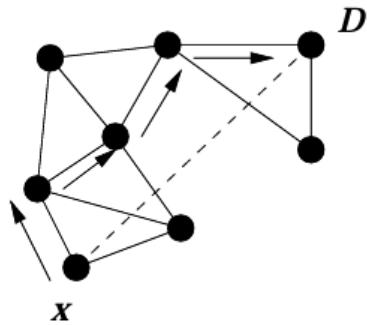


Figure 1: Esempio di instradamento nel protocollo GPSR, D è il nodo destinazione, x la sorgente

## Stato dell'Arte sugli Attacchi al Protocollo GPSR

Il protocollo **GPSR** [1] è ampiamente apprezzato per la sua scalabilità e leggerezza in scenari altamente dinamici, come quelli tipici delle FANET. Tuttavia, la mancanza di meccanismi di sicurezza integrati rende GPSR vulnerabile ad attacchi mirati, con effetti potenzialmente disastrosi sul corretto instradamento dei pacchetti.

In letteratura, gli attacchi al GPSR si concentrano principalmente su due tipologie di minacce:

- **Attacco Sybil:** In questo scenario, un nodo malizioso genera numerose identità false, introducendo nodi finti nella rete. Tale iniezione di identità fasulle può compromettere la topologia e l'affidabilità del routing, facilitando ulteriori attacchi, come il man-in-the-middle o la raccolta di informazioni sensibili [2].
- **Attacco Gray Hole:** Qui il nodo malintenzionato adotta una politica di drop selettiva, basata su criteri temporali o probabilistici, piuttosto che scartare sistematicamente tutti i pacchetti (come avviene nel Black Hole). Questa modalità di attacco rende il rilevamento più difficile, in quanto il nodo continua a inoltrare parte del traffico, mascherando la propria attività malevola e degradando progressivamente le prestazioni della rete [3].

Il contributo innovativo presentato in *SecGPSR* [4] consiste nell'introdurre una versione sicura di GPSR che integra:

- Meccanismi di autenticazione basati su crittografia a curva ellittica (ECC), utilizzando in particolare l'ECDSA per firmare i beacon di routing, al fine di garantire l'autenticità dei nodi.
- Un sistema collaborativo di valutazione della *trustness* tra i nodi, utile per identificare e isolare quelli che adottano comportamenti da Gray Hole.

Queste soluzioni sono state progettate per essere leggere e compatibili con le risorse limitate tipiche dei droni, superando le criticità delle contromisure basate su modelli complessi o approcci centralizzati [5, 6].

## 2 Gpsr con stazioni a terra: scenario ed architettura generale

Il GPSR modificato introduce **stazioni a terra** posizionate in modo equidistante in uno scenario  $n \times n$ . Ogni stazione gestisce un quadrante di dimensione:

$$l = \frac{n}{m}$$

Le stazioni mantengono una tabella di posizioni (`quadNodesPositionTable`) dei nodi nel loro quadrante e cooperano per rispondere alle richieste di posizione.

### 2.1 Messaggi

Per implementare questo nuovo paradigma sono state aggiunte nuove tipologie di messaggio oltre ai classici *beacon* di GPSR. Questi messaggi fanno riferimento alle interazioni tra nodi e stazioni e stazioni e stazioni.

Messaggio	Scopo	Mittente	Destinatario
<code>StationNotice</code>	Registrazione/Deregistrazione quadrante	Nodo	Stazione
<code>PositionRequest</code>	Richiesta posizione nodo	Nodo	Stazione
<code>PositionResponse</code>	Risposta con coordinate o fallimento	Stazione	Nodo
<code>S2SPositionRequest</code>	Richiesta posizione nodo	Stazione	Stazione
<code>S2SPositionResponse</code>	Risposta con coordinate o fallimento	Stazione	Stazione

Table 1: Tabella dei messaggi

### 2.2 Strutture dati

Oltre ai messaggi sono state aggiunte anche delle strutture dati per mantenere uno stato consistente della rete e del protocollo stesso.

Nome	<code>quadNodesPositionTable</code>	<code>destinationsPositionTable</code>	<code>targetAddressToDelayedPackets</code>
Proprietario	Stazioni a terra	Nodi mobili	Nodi mobili
Scopo	Registro nodi nel quadrante	Cache posizioni destinatari	Coda pacchetti in attesa
Formato	$\langle L3Addr, (Coord, t) \rangle$	$\langle L3Addr, (Coord, t) \rangle$	$\langle L3Addr, Queue\langle Packet \rangle \rangle$
Gestione	<ul style="list-style-type: none"> <li>Aggiornamento via <i>StationNotice</i></li> <li>Pulizia tramite <i>intervallo di validità</i></li> </ul>	<ul style="list-style-type: none"> <li>Popolamento via <i>Position-Response</i></li> <li>Pulizia tramite <i>intervallo di validità</i></li> </ul>	<ul style="list-style-type: none"> <li>Inserimento in assenza di posizione</li> <li>Svuotamento dopo <i>PositionResponse</i></li> </ul>

Table 2: Tabella delle strutture dati

**Note:**

- `quadNodesPositionTable` e `destinationsPositionTable` usano timestamp per verificare la validità delle posizioni
- `destinationsPositionTable` evita richieste ridondanti

### 2.3 Registrazione e deregistrazione dei nodi

I nodi inviano periodicamente una `StationNotice` alla stazione del quadrante in cui si trovano, all'interno della quale dichiarano la propria posizione. Nel caso in cui un nodo, spostandosi, dovesse cambiare quadrante, invia alla stazione del quadrante che sta per lasciare un messaggio di *deregistrazione* e poi si *registra* presso la stazione del nuovo quadrante.

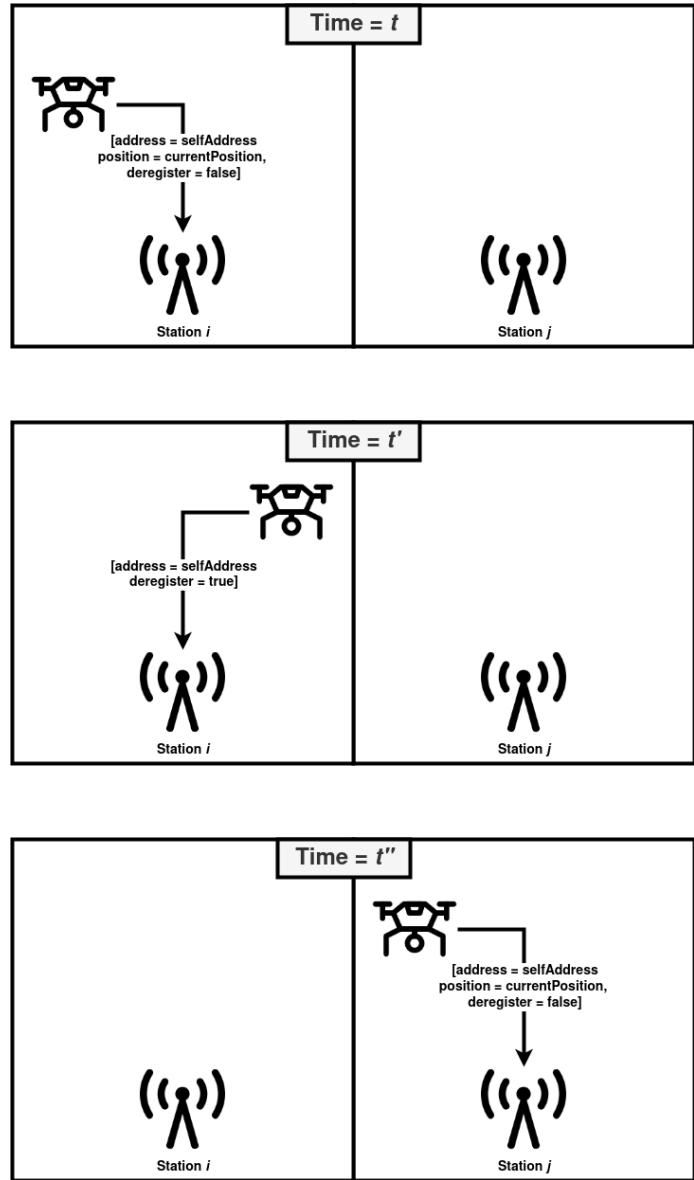


Figure 2: Esempio di registrazione e deregistrazione di un nodo

---

**Algorithm 1** Invio StationNotice da parte del nodo

---

- 1: **Class Variables:**
  - l*: dimensione quadrante
  - m*: numero stazioni per lato
  - currentStation*: indice stazione corrente
- 2: *address*  $\leftarrow$  indirizzo nodo
- 3: *pos*  $\leftarrow$  posizione corrente
- 4:  $q \leftarrow \lfloor \frac{\text{pos}_x}{l} \rfloor + m \cdot \lfloor \frac{\text{pos}_y}{l} \rfloor$
- 5: **if** *q*  $\neq$  *currentStation* **then**
- 6:     INVIA(StationNotice(*address*, deregister=true)) a *currentStation*
- 7: **end if**
- 8: INVIA(StationNotice(*address*, *pos*, deregister=false)) a *q*
- 9: *currentStation*  $\leftarrow$  *q*

---

---

**Algorithm 2** Elaborazione StationNotice da parte della stazione

---

- 1: **Input:** Messaggio StationNotice
- 2: **if** deregister == true **then**
- 3:     RIMUOVI *address* da quadNodesPositionTable
- 4: **else**
- 5:     AGGIORNA (*address*, *position*) in quadNodesPositionTable
- 6: **end if**

---

## 2.4 Richiesta ed ottenimento della posizione di un nodo

Quando un nodo deve mandare un messaggio, per il protocollo GPSR, deve conoscere la posizione del destinatario. Nella fattispecie possono verificarsi due casi:

- Il nodo ha già in cache la posizione del destinatario.
- Il nodo non conosce la posizione del destinatario, quindi deve chiedere alle stazioni.

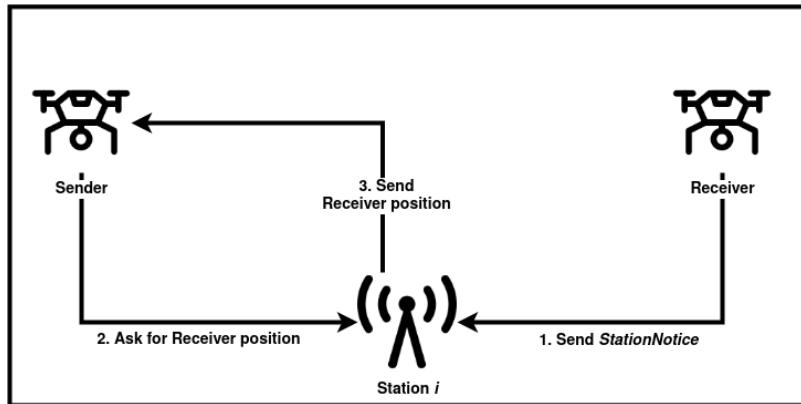


Figure 3: Richiesta posizione

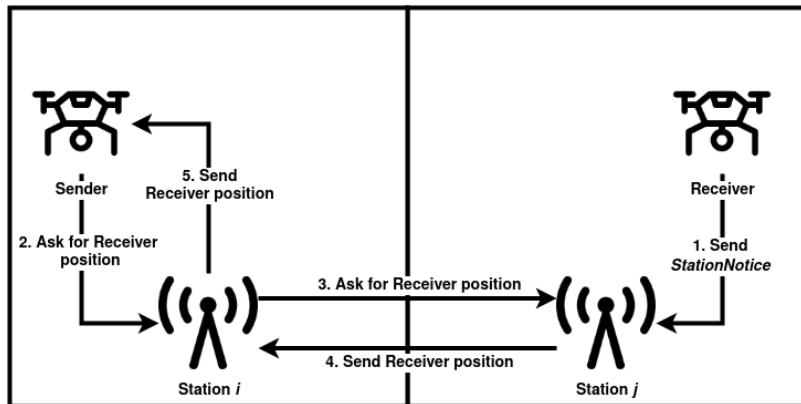


Figure 4: Richiesta posizione in un altro quadrante

---

**Algorithm 3** Gestione invio messaggio

---

- 1: **Class Variables:**  
*l*: dimensione quadrante  
*m*: numero stazioni per lato  
*currentStation*: indice stazione corrente
- 2: **Input:** destinationAddress
- 3: **if** destinationAddress non in destinationsPositionTable **then**
- 4:     INVIA(PositionRequest(destinationAddress)) a *currentStation*
- 5:     Metti il pacchetto in coda
- 6:     **return** QUEUE
- 7: **else**
- 8:     INIZIA GPSR
- 9: **end if**

---

---

**Algorithm 4** Elaborazione PositionRequest da parte di una stazione

---

- 1: **Input:** Messaggio PositionRequest
- 2: **if** destinationAddress è in quadNodesPositionTable **then**
- 3:     INVIA(PositionResponse(destinationAddress, destinationPosition)) al mittente
- 4: **else**
- 5:     INVIA(S2SPositionRequest(destinationAddress)) alle altre stazioni
- 6: **end if**

---

---

**Algorithm 5** Elaborazione PositionResponse da parte di un nodo

---

- 1: **Input:** Messaggio PositionResponse
- 2: AGGIORNA (address, position) in destinationsPositionTable
- 3: **if** ci sono messaggi in coda per destinationAddress **then**
- 4:     INVIA i messaggi con GPSR
- 5: **end if**

---

La gestione di S2SPositionRequest e S2SPositionResponse è triviale: una stazione risponde con la posizione se è presente nella sua quadNodesPositionTable altrimenti specifica nel messaggio di risposta che questa non è stata impostata, così il destinatario capisce che quella risposta non contiene alcuna posizione.

### **3 Scenario d'attacco 1: falsificazione della posizione agli altri nodi**

#### **3.1 Introduzione**

La sicurezza dell'instradamento è fondamentale per garantire prestazioni ottimali del protocollo. Un attacco significativo in questo contesto è la falsificazione delle posizioni da parte di nodi malevoli, che manipolano le informazioni nei beacon per ingannare i nodi vicini, aumentando il ritardo della rete e prolungando il tempo in modalità perimetrale. La falsificazione delle posizioni ha un impatto diretto sulle due modalità operative del GPSR:

1. Instradamento Greedy: I nodi scelgono il prossimo hop in base alla vicinanza alla destinazione. Informazioni di posizione errate possono portare alla selezione di percorsi subottimali o addirittura a loop, aumentando il ritardo complessivo.
2. Modalità Perimetrale: Quando l'instradamento greedy fallisce, il protocollo passa alla modalità perimetrale, che utilizza grafi planari per instradare i pacchetti attorno a ostacoli. La presenza di posizioni falsificate può estendere inutilmente il percorso in modalità perimetrale, incrementando il tempo di consegna dei pacchetti.

#### **3.2 Attacco**

In questo scenario, i droni malevoli falsificano deliberatamente le informazioni di posizione nei loro beacon, dichiarando coordinate errate secondo diverse strategie:

1. Falsificazione Globale: Il nodo malevolo dichiara una posizione opposta nella mappa rispetto a quella reale.
2. Falsificazione Locale: Il nodo dichiara di trovarsi nella posizione opposta a quella reale nel suo quadrante di appartenenza.
3. Falsificazione Casuale: Il nodo fornisce coordinate completamente casuali.

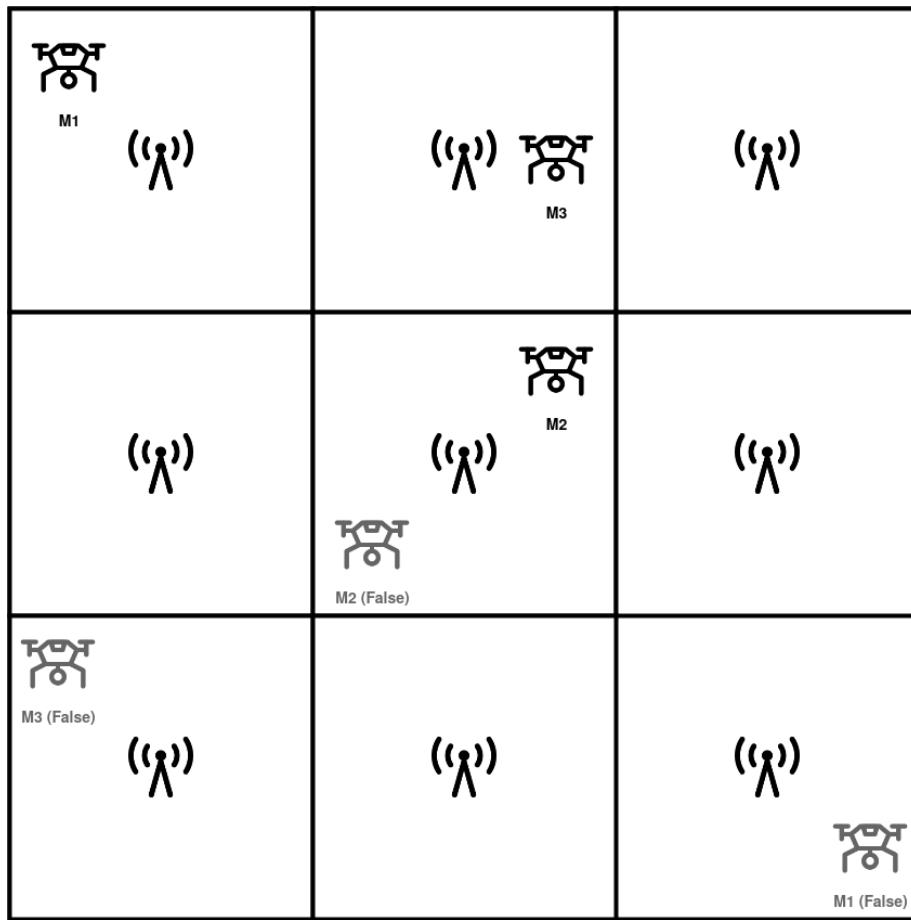


Figure 5: Esempio di generazione di posizioni false

Nel protocollo GPSR, i nodi trasmettono periodicamente beacon contenenti il proprio indirizzo e le coordinate attuali, permettendo ai nodi vicini di aggiornare le loro tabelle di vicinato (`neighboursPositionTable`) . I nodi malevoli alterano questi beacon inserendo informazioni di posizione falsificate secondo le strategie sopra descritte. Questa manipolazione induce i nodi vicini a registrare posizioni errate nelle loro tabelle, compromettendo l'efficacia dell'instradamento.

### 3.3 Implementazione

---

**Algorithm 6** Generazione Posizione Falsa

---

**Require:** Posizione reale  $(x, y, z)$ , parametri  $n, m, z_{min}, z_{max}$

**Ensure:** Posizione falsa  $(x', y', z')$

```

1:  $\Delta \leftarrow n/m$ 
2:  $q_x \leftarrow \lfloor x/\Delta \rfloor$ 
3:  $q_y \leftarrow \lfloor y/\Delta \rfloor$ 
4:  $r \leftarrow \text{random}(0, 1)$ 
5: if  $r \leq 0.33$  then
6:   Strategia 1: Falsificazione Globale
7:    $x' \sim U(0, \Delta) \cup U(n - \Delta, n)$ 
8:    $y' \sim U(0, \Delta) \cup U(n - \Delta, n)$ 
9: else if  $0.33 < r \leq 0.66$  then
10:  Strategia 2: Falsificazione locale
11:   $c_x \leftarrow (q_x + 0.5)\Delta$ 
12:   $c_y \leftarrow (q_y + 0.5)\Delta$ 
13:   $x' \leftarrow 2c_x - x$ 
14:   $y' \leftarrow 2c_y - y$ 
15: else
16:  Strategia 3: Falsificazione Casuale
17:   $x' \sim U(0, n)$ 
18:   $y' \sim U(0, n)$ 
19: end if
20:  $z' \sim U(z_{min}, z_{max})$ 
21: return  $(x', y', z')$ 

```

---



---

**Algorithm 7** Creazione Beacon

---

```

1: address  $\leftarrow$  indirizzo nodo
2: realPos  $\leftarrow$  GETREALPOSITION
3: if isMalicious then
4:   pos  $\leftarrow$  GENERATEFALSEPOSITION(realPos)
5: else
6:   pos  $\leftarrow$  realPos
7: end if
8: return Beacon(address, pos)

```

---

### 3.4 Valutazione delle performance

Per analizzare l'impatto dell'attacco da parte di nodi malevoli sul protocollo, sono state condotte simulazioni in OMNeT++ in un'area di  $1000 \times 1000$  metri, con un totale di 100 nodi. Il numero di nodi malevoli è stato variato da 0 a 40, al fine di evidenziare come l'inserimento di comportamenti malevoli influenzino le prestazioni complessive del protocollo. Le metriche analizzate permettono di valutare in che modo il falsamento della posizione verso i nodi vicini possa degradare il tempo di consegna dei pacchetti, aumentando l'overhead del routing e il tempo in perimeter time.

I parametri di simulazione adottati sono riportati nella Tabella:

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di nodi malevoli	0
Numero massimo di nodi malevoli	40
Tempo di simulazione	60/120 s
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 3: Parametri della simulazione

### 3.4.1 60 secondi

I grafici presentati in questa sezione sono relativi alla modalità di invio *Interval*, che rispecchia in maniera più realistica il comportamento delle reti wireless, dove l'invio dei pacchetti risulta variabile nel tempo anziché essere fisso.

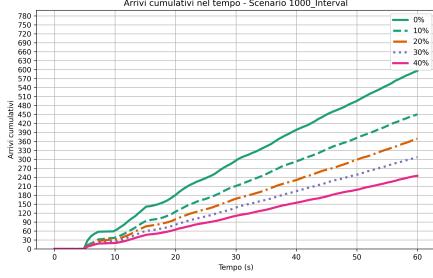


Figure 6: Arrivi cumulativi nel tempo (60 s)

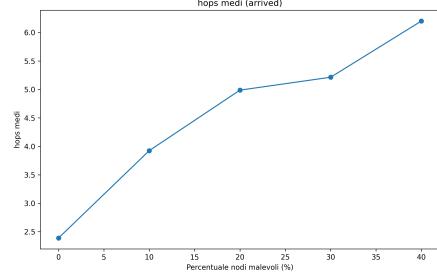


Figure 7: Mean Hops (60 s)

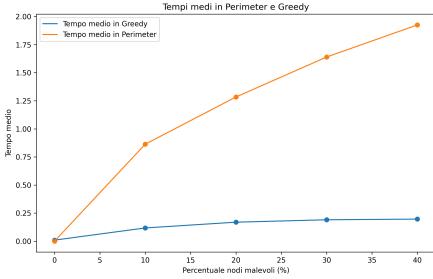


Figure 8: Per Greedy Time (60 s)

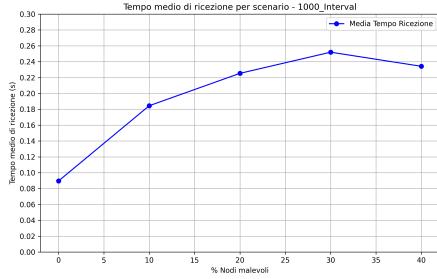


Figure 9: Reception Time (60 s)

In particolare, i grafici illustrano l'impatto crescente dei nodi malevoli, variando il loro numero da 0 a 40. Dalle simulazioni emerge che, al crescere del numero di nodi malevoli, il numero di messaggi correttamente ricevuti diminuisce costantemente. Contestualmente, si osserva un aumento del numero medio di hop necessari per raggiungere il destinatario, segno di percorsi meno diretti ed efficienti. Inoltre, il tempo trascorso in modalità *perimeter* (espresso nel grafico relativo al *perimeter greedy time*) e il tempo medio di ricezione dei messaggi aumentano, indicando una maggiore latenza nella consegna dei pacchetti. Queste evidenze dimostrano come l'incremento dei nodi malevoli influisca negativamente sulle performance del protocollo, degradando sia l'efficienza del percorso che i tempi di consegna.

### 3.4.2 120 secondi

In questa sezione vengono analizzati i risultati ottenuti estendendo il tempo di simulazione a 120 secondi, il doppio rispetto alle precedenti simulazioni. Questo incremento temporale consente di osservare con maggiore chiarezza le tendenze già evidenziate nei grafici a 60 secondi, permettendo un'analisi più approfondita sull'evoluzione delle prestazioni del protocollo in presenza di nodi malevoli.

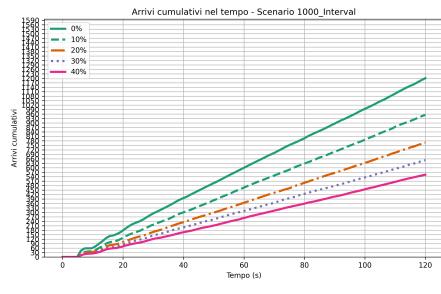


Figure 10: Arrivi cumulativi nel tempo (120 s)

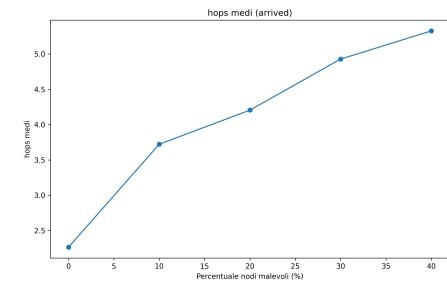


Figure 11: Mean Hops (120 s)

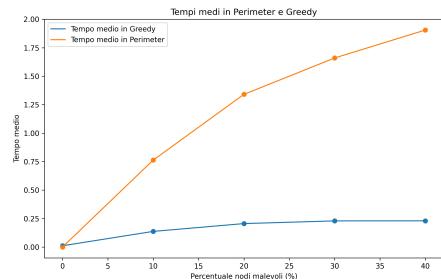


Figure 12: Per Greedy Time (120 s)

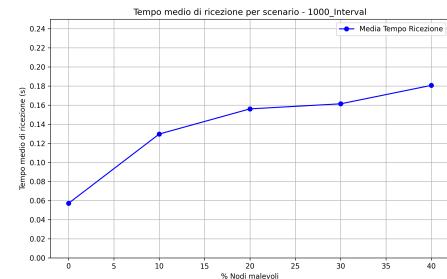


Figure 13: Reception Time (120 s)

Come prevedibile, il numero complessivo di messaggi inviati e ricevuti risulta maggiore, data la durata estesa della simulazione. Tuttavia, l'effetto negativo introdotto dai nodi malevoli rimane invariato: il numero di pacchetti consegnati con successo diminuisce progressivamente all'aumentare del numero di attaccanti, mentre il numero medio di hop necessari per raggiungere la destinazione continua ad aumentare.

Allo stesso modo, il tempo trascorso in modalità *perimeter* (mostrato nel grafico *perimeter greedy time*) e il tempo medio di ricezione dei pacchetti subiscono un incremento, confermando che la presenza di nodi malevoli compromette la rapidità e l'efficienza del routing. La maggiore durata della simulazione amplifica questi effetti, evidenziando con ancora più chiarezza il degrado delle prestazioni del protocollo al crescere degli attacchi nella rete.

## 4 L'utilizzo della crittografia

### 4.1 Introduzione

La mitigazione del problema delle posizioni falsificate, in questo scenario, avviene attraverso l'adozione di meccanismi di crittografia asimmetrica basata su curve ellittiche [4], che consentono di garantire l'autenticità e l'integrità delle informazioni sulla posizione trasmesse nei beacon. In particolare, vengono impiegati algoritmi quali ECDSA e EDDSA per la firma digitale, noti per la loro efficienza e sicurezza.

### 4.2 Crittografia asimmetrica e curve ellittiche

La crittografia asimmetrica (o a chiave pubblica) utilizza una coppia di chiavi correlate matematicamente: una chiave privata, mantenuta segreta, e una chiave pubblica, diffusa liberamente. L'uso delle curve ellittiche (ECC, Elliptic Curve Cryptography) offre il vantaggio di ottenere un elevato livello di sicurezza con chiavi di dimensioni notevolmente inferiori rispetto ad altri sistemi (ad esempio RSA). Ciò risulta particolarmente utile in applicazioni dove le risorse computazionali e di memoria sono limitate, come nel caso dei droni.

Elliptic Curve Cryptography (ECC) è un sistema di crittografia asimmetrica che sfrutta la struttura algebrica delle curve ellittiche per la generazione di chiavi pubbliche e private. Una curva ellittica, in forma generale, è definita da un'equazione del tipo:

$$y^2 = x^3 + ax + b, \quad a, b \in \mathbb{R}$$

La condizione di non-singolarità:

$$4a^3 + 27b^2 \neq 0$$

garantisce che la curva non presenti punti critici. In applicazioni crittografiche, la curva viene definita su un campo finito  $\mathbb{F}_p$ :

$$y^2 \equiv x^3 + ax + b \pmod{p}, \quad a, b \in \mathbb{F}_p, \quad p \text{ è un numero primo}$$

La forza di ECC risiede nell'efficiente generazione di chiavi: dato un punto  $P$  sulla curva, la chiave pubblica viene ottenuta calcolando:

$$X = x \cdot P$$

dove  $x$  è un intero casuale scelto come chiave privata. La sicurezza dell'algoritmo è basata sulla difficoltà di risolvere il problema del logaritmo discreto sulle curve ellittiche (ECDLP), che permette di utilizzare chiavi di dimensioni ridotte mantenendo elevati livelli di sicurezza.

In particolare, ECC può essere utilizzato come altri schemi di crittografia asimmetrica per calcolare la firma digitale, in modo tale da garantire non solo la confidenzialità ma anche l'integrità e il non ripudio dei dati trasmessi. L'algoritmo di firma digitale (DSA) esteso alla crittografia a curve ellittiche è chiamato **ECDSA**. Esso permette di generare e verificare firme basandosi sulla sicurezza garantita dall'ECDLP.

**EDDSA** (Edwards-curve Digital Signature Algorithm) è un algoritmo di firma digitale che rappresenta una variante moderna rispetto a ECDSA, ma che sfrutta anch'esso la crittografia basata su curve ellittiche. Mentre ECDSA opera generalmente su curve ellittiche nella forma di Weierstrass, EDDSA utilizza curve di tipo Edwards, che offrono diversi vantaggi a livello prestazionale e di sicurezza. Infatti, EDDSA è progettato per essere deterministico, eliminando la necessità di un valore casuale diverso per ogni firma. Questo si traduce in operazioni di firma e verifica molto veloci, rendendo l'algoritmo particolarmente adatto per dispositivi con risorse limitate e applicazioni ad alto throughput come i droni.

## 5 Mitigazione per falsificazione della posizione agli altri nodi (Scenario d'attacco 1)

### 5.1 Introduzione

Si introduce un meccanismo di autenticazione crittografica per mitigare il problema della falsificazione delle posizioni nei beacon, integrandola con una gestione dinamica della fiducia (trust management). Le stazioni vengono dotate di una coppia di chiavi pubbliche e private ECDSA o EDDSA, a seconda dell'algoritmo si scelga di utilizzare per la firma digitale.

### 5.2 Meccanismo di mitigazione

La mitigazione comprende diverse fasi, l'introduzione di un nuovo messaggio, `StationNoticeResponse` e una modifica al messaggio di GPSR Beacon.

Messaggio	Scopo	Mittente	Destinazione
Station Notice Response	Restituisce la posizione, la sua firma ed il nonce	Stazione	Nodo

Table 4: Descrizione del messaggio Station Notice Response

Le fasi principali sono:

1. **Invio della `stationNotice`:** Ogni nodo invia periodicamente un messaggio `stationNotice` contenente:
  - Il proprio indirizzo.
  - La posizione dichiarata.
  - flag di registrazione.
2. **Elaborazione dei dati da parte della stazione:** La stazione verifica l'identità del nodo e lo aggiunge alla sua tabella delle posizioni `quadNodesPositionTable`. La stazione inoltre:
  - Genera un nonce.
  - Firma la coppia posizione-nonce con la propria chiave privata utilizzando un algoritmo crittografico sicuro (ECDSA o EDDSA).
  - Restituisce al nodo corrispondente il messaggio di (`StationNoticeResponse`), contenente la posizione dichiarata, la firma della posizione ed il nonce utilizzato.
3. **Invio del messaggio di GPSR Beacon ai vicini:** I nodi inviano ai vicini un messaggio di Beacon modificato rispetto all'originale, contenente:
  - Indirizzo del nodo.
  - Posizione restituita dalla `StationNoticeResponse`.
  - Firma della posizione dichiarata.
  - Nonce utilizzato per la firma.

**4. Verifica della firma e aggiornamento tabella dei vicini:** All'arrivo dei messaggi di Beacon, i nodi verificano la firma contenuta nel beacon utilizzando la chiave pubblica della stazione corrispondente:

- Qualora l'esito sia positivo, il nodo aggiunge l'indirizzo contenuto nel Beacon nella propria tabella dei vicini.
- Altrimenti, il messaggio viene scartato e viene applicata una penalità associata all'indirizzo del mittente del Beacon.

### 5.3 Elaborazione dei dati da parte della stazione

La funzione `processStationNotice` implementa la logica di elaborazione dei messaggi `stationNotice` all'interno della stazione. Oltre all'aggiornamento della `quadNodesPositionTable`, viene creato il messaggio di `StationNoticeResponse` contenente la firma della posizione dichiarata nella `StationNotice`.

### 5.4 Dettagli dell'algoritmo `processStationNotice`

L'algoritmo `processStationNotice` ha lo scopo di gestire i messaggi di notifica inviati da una stazione, aggiornando la tabella delle posizioni dei nodi e rispondendo con un messaggio certificato. Di seguito vengono descritti i dettagli operativi:

**Input** L'algoritmo riceve come input un pacchetto contenente le seguenti informazioni:

- **Indirizzo sorgente:** l'identificativo del nodo che ha inviato la richiesta.
- **Posizione:** la posizione del nodo che ha inviato il messaggio.
- **Flag di deregistrazione:** un indicatore che specifica se il nodo vuole essere rimosso dalla tabella delle posizioni.

**Struttura dell'Algoritmo** L'algoritmo segue i seguenti passi:

1. **Estrazione delle informazioni:** Il pacchetto viene analizzato per ottenere i dati essenziali, ovvero l'indirizzo del nodo e la sua posizione.
2. **Gestione della deregistrazione:**
  - Se il nodo ha richiesto la rimozione (flag di deregistrazione attivo), si verifica se la sua posizione è registrata nella `quadNodesPositionTable`.
  - Se la posizione è presente, viene rimossa dalla tabella.
3. **Registrazione della posizione:** Se il nodo non ha richiesto la deregistrazione, la sua posizione viene aggiornata nella tabella delle posizioni.
4. **Creazione della risposta:** Viene generato un messaggio di risposta con la posizione certificata.
5. **Invio della risposta:** Il messaggio di risposta viene inviato al nodo che ha originato la richiesta.

**Output** L'algoritmo non restituisce un valore diretto, ma aggiorna la tabella delle posizioni e invia un messaggio di risposta al nodo sorgente. In caso di deregistrazione, il nodo viene rimosso dalla tabella.

---

**Algorithm 8** processStationNotice

---

```

1: function PROCESSSTATIONNOTICE(req)
2:   address ← req.getSource()
3:   position ← req.getPosition()
4:   if req.isDeregister() then
5:     if quadNodesPositionTable.hasPosition(address) then
6:       quadNodesPositionTable.removePosition(address)
7:     end if
8:   else
9:     quadNodesPositionTable.setPosition(address, position)
10:    response ← createStationNoticeResponse(position.x, position.y, position.z)
11:    sendStationNoticeResponse(response, req.getSource())
12:  end if
13: end function
```

---

## 5.5 Dettagli dell'algoritmo *createStationNoticeResponse*

L'algoritmo `createStationNoticeResponse` ha lo scopo di generare un messaggio di risposta contenente una posizione certificata. Questo messaggio viene firmato digitalmente per garantire autenticità e integrità.

**Input** L'algoritmo riceve in input:

- **Coordinate**  $(x, y, z)$ : la posizione del nodo che deve essere certificata.

**Struttura dell'Algoritmo** L'algoritmo esegue i seguenti passi:

1. **Creazione della risposta:** Viene istanziato un nuovo oggetto di tipo `StationNoticeResponse`.
2. **Creazione della stringa:** La posizione viene concatenata in una stringa nel formato " $x-y-z$ ".
3. **Generazione del nonce:** Viene generato un valore casuale (*nonce*) per proteggere l'integrità del messaggio, identificandolo univocamente.
4. **Firma digitale:**
  - Se l'algoritmo di firma è **ECDSA**, la firma viene calcolata usando `signMessageECDSA()`.
  - Altrimenti, viene utilizzato **EDDSA** con `signMessageEDDSA()`.
5. **Assegnazione dei valori:** La risposta viene aggiornata con la posizione, il nonce e la firma digitale.
6. **Restituzione del messaggio:** Il messaggio firmato viene restituito come output.

**Output** L'algoritmo restituisce un oggetto **StationNoticeResponse**, contenente:

- **Firma digitale** generata con ECDSA o EDDSA.
- **Coordinate**  $(x, y, z)$  della posizione certificata.
- **Nonce**.
- **Lunghezza totale del messaggio**.

---

**Algorithm 9** createStationNoticeResponse

---

```
1: function CREATESTATIONNOTICERESPONSE(x, y, z)
2:   stationResponse  $\leftarrow$  new StationNoticeResponse()
3:   clear  $\leftarrow$  convertToString(x) + "-" + convertToString(y) + "-" + convertToString(z)
4:   nonce  $\leftarrow$  nonceGen.generateNonce()
5:   signature  $\leftarrow$  ""
6:   if signAlgorithm == ECDSA then
7:     signature  $\leftarrow$  signMessageECDSA(keyPairECDSA, clear, nonce)
8:   else
9:     signature  $\leftarrow$  signMessageEDDSA(clear, privateKeyEDDSA, nonce)
10:  end if
11:  stationResponse.setC(signature)
12:  stationResponse.setX(x)
13:  stationResponse.setY(y)
14:  stationResponse.setZ(z)
15:  stationResponse.setNonce(nonce)
16:  return stationResponse
17: end function
```

---

## 5.6 Dettagli dell'algoritmo *createBeacon*

L'algoritmo `createBeacon` genera un messaggio di beacon che viene utilizzato nel protocollo GPSR per la comunicazione tra i nodi della rete. Il comportamento dell'algoritmo varia in base al fatto che il nodo sia **onesto** o **malevolo**.

**Input** L'algoritmo prende in ingresso i seguenti parametri:

- **position**: La posizione da includere nel beacon, ottenuta dal messaggio di `StationNoticeResponse`.
- **signature**: Firma digitale della posizione del nodo per garantirne l'autenticità.
- **nonce**.

**Struttura dell'Algoritmo** L'algoritmo segue i seguenti passi:

1. **Creazione del beacon**: Viene istanziato un nuovo oggetto di tipo `GpsrBeacon`.
  - Se il nodo è **malevolo**, vengono assegnati:
    - **Indirizzo reale**: L'indirizzo del beacon è quello del nodo attuale.
    - **Posizione falsificata**: Viene generata una posizione errata con `generateFalsePosition()`.
  - Se il nodo è **onesto**, vengono assegnati:
    - **Indirizzo reale**: L'indirizzo del beacon è quello del nodo attuale.
    - **Posizione reale**: La posizione è ottenuta dal messaggio di `StationNoticeResponse`.
2. **Firma e Nonce**: In entrambi i casi, viene inclusa la firma della posizione e il nonce ricevuto in input.
3. **Restituzione del beacon**: L'oggetto `GpsrBeacon` viene restituito come output.

---

**Algorithm 10** createBeacon

---

```
1: function CREATEBEACON(position, signature, nonce)
2:   beacon ← new GpsrBeacon()
3:   if isMalicious then
4:     beacon.address ← getSelfAddress()
5:     beacon.position ← generateFalsePosition()
6:     beacon.signature ← signature
7:     beacon.nonce ← nonce
8:   else
9:     beacon.address ← getSelfAddress()
10:    beacon.position ← position
11:    beacon.signature ← signature
12:    beacon.nonce ← nonce
13:   end if
14:   return beacon
15: end function
```

---

## 5.7 Meccanismo di trustness

Per garantire l'affidabilità delle informazioni scambiate tra i nodi all'interno della rete, viene implementato un meccanismo di trustness basato sulla verifica delle firme digitali e sull'assegnazione di penalità ai nodi ritenuti inaffidabili.

**Definizione degli insiemi fondamentali** Sia  $\mathcal{N}$  l'insieme dei nodi della rete e  $\mathcal{S}$  l'insieme delle stazioni di riferimento. Ogni stazione  $S_k \in \mathcal{S}$  possiede una coppia di chiavi crittografiche ( $\text{PUB}_k, \text{PRIV}_k$ ), rispettivamente la chiave pubblica e la chiave privata. La chiave pubblica di ogni stazione è nota a tutti i nodi della rete.

**Beacon e Posizioni** Ogni nodo  $N_i \in \mathcal{N}$  trasmette periodicamente un messaggio di beacon contenente la propria posizione dichiarata  $P_i$ . Un nodo malevolo potrebbe trasmettere una posizione falsa  $P'_i$  al fine di manipolare il protocollo di routing.

Un beacon trasmesso da  $N_i$  è quindi definito come:

$$B_i = (A_i, P_i, \text{Signature}_i, \text{Nonce}_i) \quad (3)$$

dove:

- $A_i$  è l'indirizzo del nodo mittente;
- $P_i$  è la posizione dichiarata;
- $\text{Signature}_i$  è la firma digitale generata con la chiave privata della stazione di riferimento;
- $\text{Nonce}_i$  è un valore casuale utilizzato per la firma e per prevenire attacchi di replay.

## 5.8 Calcolo del valore di trustness

Quando un nodo  $N_j$  riceve il beacon  $B_i$  dal nodo  $N_i$ , esegue una verifica della firma  $\text{Signature}_i$  utilizzando la chiave pubblica della stazione di riferimento. Tale verifica assicura che la posizione dichiarata  $P_i$  corrisponda a quella firmata e non sia stata alterata da terzi.

Per ogni nodo  $N_j$ , vicino a  $N_i$ , definiamo la variabile booleana:

$$\text{verified} \in \{\text{true}, \text{false}\} \quad (4)$$

La verifica viene eseguita tramite la funzione:

$$\text{verified} = \text{VerifyAlg}(\text{PUB}_s, \text{Signature}_i, \text{Nonce}_i, P_i) \quad (5)$$

Se la verifica fallisce, il nodo  $N_j$  applica una penalità al livello di fiducia (trustness) associato all'indirizzo  $\text{Addr}_i$ . Indichiamo con  $\text{Trust}(\text{Addr}_i)$  il valore corrente di fiducia del nodo  $N_i$ . La penalità viene applicata attraverso la seguente procedura:

1. Calcolo del valore penalizzato:

$$\text{measurement} = \text{Trust}(\text{Addr}_i) - \text{PENALTY} \quad (6)$$

2. Aggiornamento della fiducia tramite meccanismo di smoothing:

$$\text{Trust}(\text{Addr}_i) = (1 - \alpha) \cdot \text{Trust}(\text{Addr}_i) + \alpha \cdot \text{measurement}, \quad \alpha \in [0, 1] \quad (7)$$

3. Se  $\text{Trust}(\text{Addr}_i)$  scende al di sotto della soglia  $\text{notTrusted}$ , il nodo  $N_i$  viene marcato come malevolo e i suoi beacon sono sempre scartati.
4. Se la verifica ha esito positivo, la fiducia del nodo aumenta seguendo lo stesso procedimento.

$$\text{measurement} = \text{Trust}(\text{Addr}_i) + \text{PENALTY} \quad (8)$$

$$\text{Trust}(\text{Addr}_i) = (1 - \alpha) \cdot \text{Trust}(\text{Addr}_i) + \alpha \cdot \text{measurement}, \quad \alpha \in [0, 1] \quad (9)$$

5. Se  $\text{Trust}(\text{Addr}_i)$  supera la soglia  $\text{trusted}$ , i beacon del nodo vengono sempre accettati evitando ulteriori controlli.

## 5.9 Lo smoothing

Il meccanismo di smoothing evita cambiamenti bruschi, rendendo il sistema più stabile. Esso aiuta a ridurre l'impatto di una singola misurazione errata e a rendere il sistema più robusto contro falsi positivi o variazioni improvvise.

## 5.10 Algoritmo di elaborazione dei beacon

### Input:

- *packet*: pacchetto contenente il beacon ricevuto.
- *trustness*: tabella contenente i livelli di fiducia associati agli indirizzi.
- *notTrusted*: soglia al di sotto della quale un nodo è considerato non affidabile.
- *trusted*: soglia al di sopra della quale un nodo è considerato affidabile.
- *keyPairStationMapECDSA, publicKeyMapEDDSA*: mappe delle chiavi pubbliche delle stazioni utilizzate per la verifica della firma.
- $\alpha$ : parametro di smoothing per l'aggiornamento della fiducia.
- *PENALTY*: valore di penalità applicato in caso di verifica fallita.

### Output:

- Aggiornamento della tabella dei vicini.
- Aggiornamento della fiducia associata al nodo trasmittente.
- Possibile classificazione di un nodo come malevolo.

### Descrizione dell'Algoritmo:

1. Estrazione del beacon dal pacchetto ricevuto.

2. Identificazione dell'indirizzo sorgente *beaconAddress*.
3. Se il livello di fiducia di *beaconAddress* è inferiore a *notTrusted*:
  - Terminazione dell'algoritmo.
4. Se il livello di fiducia di *beaconAddress* è superiore a *trusted*:
  - La posizione del nodo viene aggiornata nella tabella dei vicini.
  - Terminazione dell'algoritmo.
5. Se il nodo è in una zona di fiducia intermedia:
  - Estrazione della posizione *pos* e del nonce *nonce*.
  - Creazione della stringa *clear* con le coordinate del nodo.
  - Identificazione della stazione di riferimento *stationIndex*.
  - Verifica della firma:
    - Se l'algoritmo di firma è ECDSA, si usa *keyPairStationMapECDSA[stationIndex]*.
    - Altrimenti, viene utilizzata la chiave *publicKeyMapEDDSA[stationIndex]*.
  - Se la verifica della firma ha successo:
    - Il livello di fiducia viene incrementato.
    - La posizione del nodo viene aggiornata nella tabella dei vicini.
  - Altrimenti:
    - Il livello di fiducia viene decrementato.

## 5.11 Implementazione

---

**Algorithm 11** Processamento del Beacon

---

```
1: function PROCESSBEACON(beacon)
2:   beaconAddress ← beacon.getAddress()
3:   if trustness[beaconAddress] ≤ notTrusted then
4:     return
5:   end if
6:   if trustness[beaconAddress] ≥ trusted then
7:     neighborPositionTable.setPosition(beacon.getAddress(), beacon.getPosition())
8:     return
9:   end if
10:  pos ← beacon.getPosition()
11:  nonce ← beacon.getNonce()
12:  clear ← convertToString(pos.x) + " - " + convertToString(pos.y) + " - " + convert-
    ToString(pos.z)
13:  stationIndex ← getStationIndex(pos)
14:  signature ← beacon.getSignature()
15:  verified ← False
16:  if signAlgorithm == ECDSA then
17:    verified ← verifySignatureECDSA(keyPairStationMapECDSA[stationIndex], clear,
    nonce, signature)
18:  else
19:    verified ← verifySignatureEDDSA(clear, signature, publicKeyMapED-
    DSA[stationIndex], nonce)
20:  end if
21:  if verified then
22:    measurement ← trustness[beaconAddress] + PENALTY
23:  else
24:    measurement ← trustness[beaconAddress] - PENALTY
25:  end if
26:  trustness[beaconAddress] ← (1 - α) · trustness[beaconAddress] + α · measurement
27: end function
```

---

## 5.12 Valutazione delle performance

Per valutare l'efficacia del metodo di mitigazione proposto, sono state condotte simulazioni su OMNeT++ considerando un'area di  $1000 \times 1000$  m con un numero totale di 100 nodi. Durante le simulazioni, il numero di nodi malevoli varia da 0 a 40, permettendo di analizzare gli effetti delle misure di mitigazione adottate. Le prestazioni del sistema sono state esaminate attraverso diverse metriche, che verranno illustrate nelle sezioni seguenti.

I parametri di simulazione adottati sono riportati nella tabella:

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di nodi malevoli	0
Numero massimo di nodi malevoli	40
Tempo di simulazione	60/120 s
Algoritmo di firma	ECDSA / EDDSA
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 5: Parametri della simulazione

### 5.12.1 60 secondi

In questa sezione vengono confrontati i risultati ottenuti in presenza dell'attacco con quelli ottenuti applicando la strategia di mitigazione. Oltre alle metriche già analizzate, viene introdotta una nuova misura relativa al consumo energetico dei droni, al fine di valutare l'impatto delle contromisure adottate.

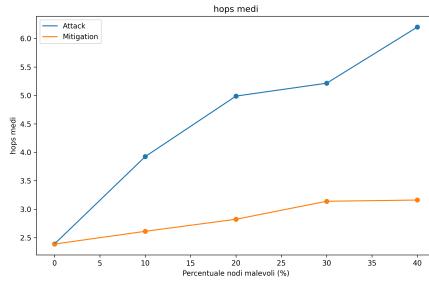


Figure 14: Mean Hops (60 s)

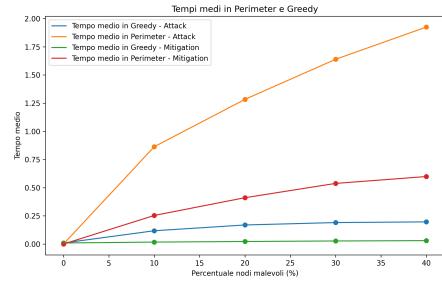


Figure 15: Per Greedy Time (60 s)

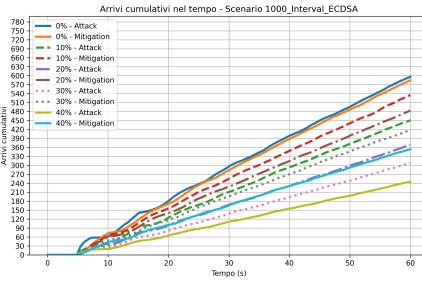


Figure 16: Arrivi cumulativi nel tempo (60 s)

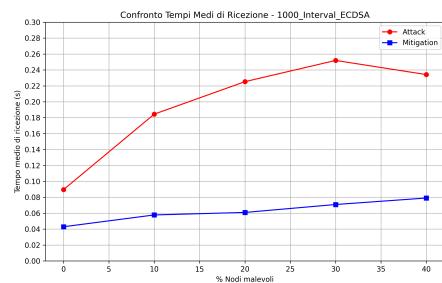


Figure 17: Reception Time (60 s)

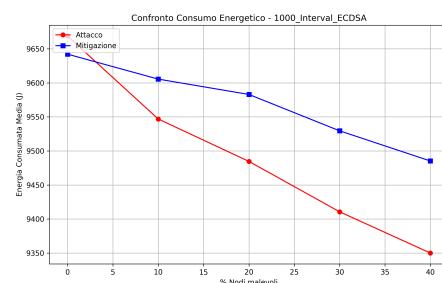


Figure 18: Consumo di energia (60 s)

Utilizzando l'algoritmo di firma EDDSA:

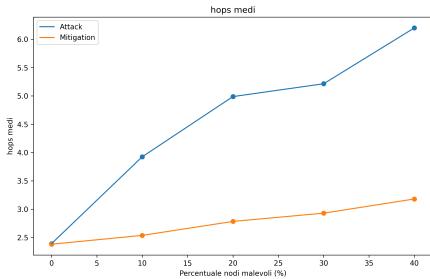


Figure 19: Mean Hops (60 s)

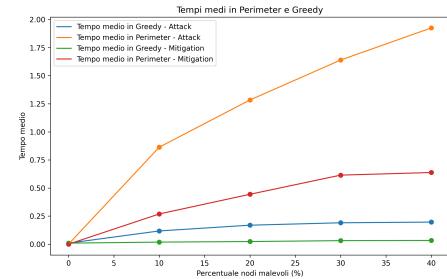


Figure 20: Per Greedy Time (60 s)

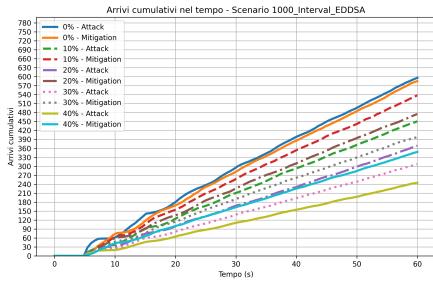


Figure 21: Arrivi cumulativi nel tempo (60 s)

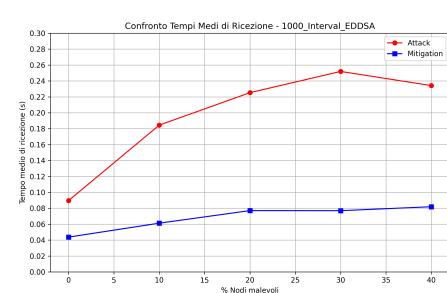


Figure 22: Reception Time (60 s)

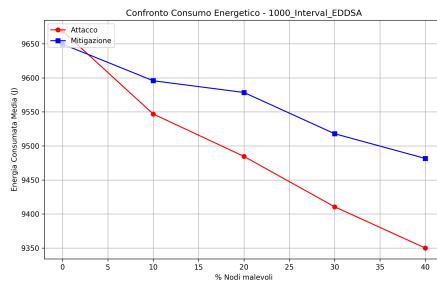


Figure 23: Consumo di energia (60 s)

L'analisi dei grafici evidenzia chiaramente come l'adozione delle mitigazioni migliori significativamente le prestazioni del protocollo:

- Maggiore numero di pacchetti ricevuti:** l'esclusione dei nodi malevoli dalle liste dei vicini consente di ripristinare percorsi di instradamento più affidabili, incrementando la probabilità di consegna dei pacchetti.
- Numero medio di hop:** La mitigazione consente di raggiungere la destinazione con un

numero di hop inferiore rispetto all'attacco.

- **Riduzione del tempo medio di ricezione:** la mitigazione riduce drasticamente il ritardo nella consegna dei pacchetti, permettendo una comunicazione più efficiente.
- **Diminuzione del tempo in modalità *perimeter*:** la corretta gestione delle liste dei vicini impedisce ai pacchetti di essere deviati su percorsi subottimali, riducendo la necessità di attivare il *perimeter forwarding*.

Questi benefici si osservano in entrambe le modalità di firma digitale, sia con **ECDSA** che con **EDDSA**, confermando l'efficacia del meccanismo di mitigazione indipendentemente dall'algoritmo crittografico adottato.

In contrapposizione, si evince un aumento dei consumi dovuto alle attività di cifratura e de-cifratura dei nodi che consumano molta energia

### 5.12.2 120 secondi

L'estensione del tempo di simulazione a 120 secondi conferma i benefici osservati con le mitigazioni. Anche in questo scenario, si registra un aumento del numero di pacchetti consegnati, una riduzione significativa del tempo medio di ricezione, del tempo trascorso in modalità *perimeter* e del numero medio di hop per raggiungere la destinazione. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano la stabilità e l'efficacia del protocollo di mitigazione su periodi di simulazione più lunghi.

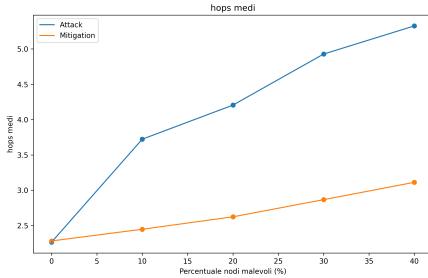


Figure 24: Mean Hops (120 s)

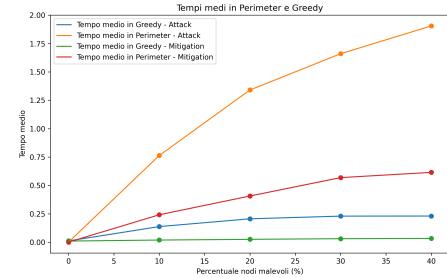


Figure 25: Per Greedy Time (120 s)

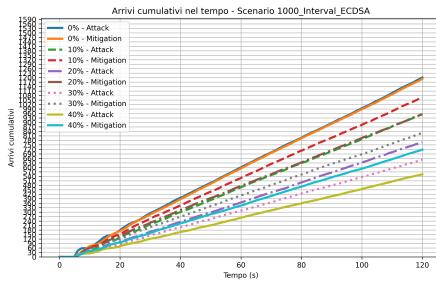


Figure 26: Arrivi cumulativi nel tempo (120 s)

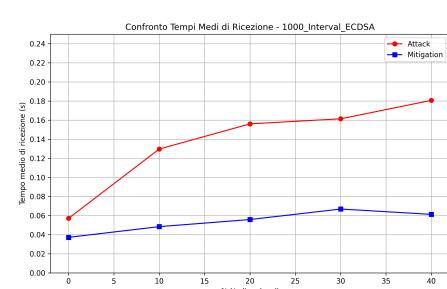


Figure 27: Reception Time (120 s)

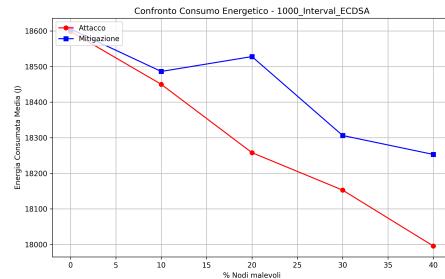


Figure 28: Consumo di energia (120 s)

Utilizzando l'algoritmo di firma EDDSA:

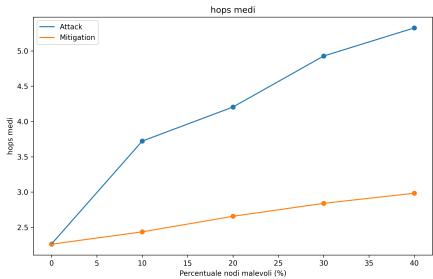


Figure 29: Mean Hops (120 s)

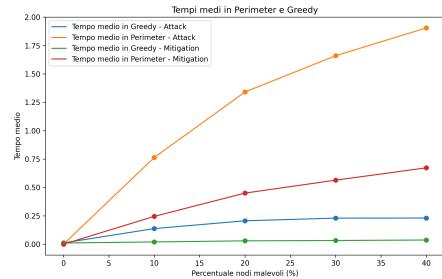


Figure 30: Per Greedy Time (120 s)

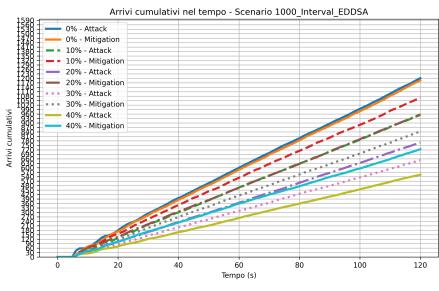


Figure 31: Arrivi cumulativi nel tempo (120 s)

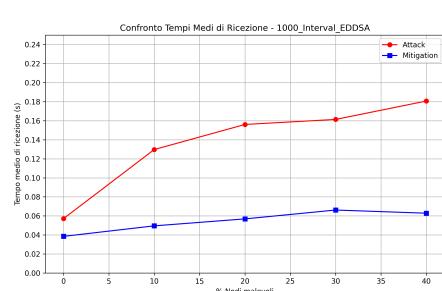


Figure 32: Reception Time (120 s)

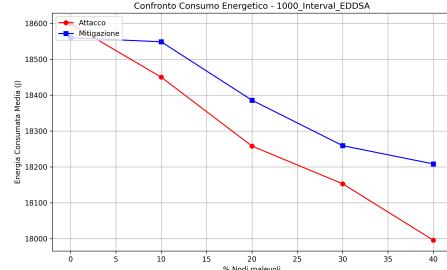


Figure 33: Consumo di energia (120 s)

Tuttavia, il consumo energetico risulta maggiore rispetto allo scenario senza mitigazioni, poiché i nodi eseguono operazioni aggiuntive per il controllo e l'esclusione dei nodi malevoli. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano l'efficacia del protocollo di mitigazione, pur con un costo energetico aggiuntivo.

### 5.12.3 Confronto fra ECDSA e EDDSA 60 secondi

In questa sezione viene analizzato il confronto tra le due modalità di firma digitale adottate, **ECDSA** ed **EDDSA**, valutandone l'impatto sulle metriche considerate. Oltre alle metriche già esaminate, viene introdotta la misura del *tempo medio di firma e verifica*, al fine di quantificare le differenze in termini di efficienza computazionale tra i due algoritmi.

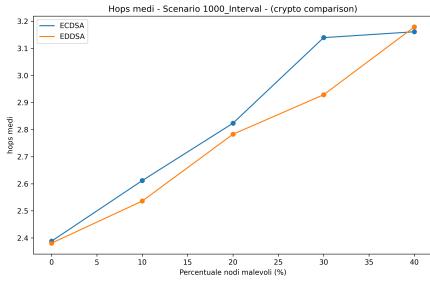


Figure 34: Mean Hops (60 s)

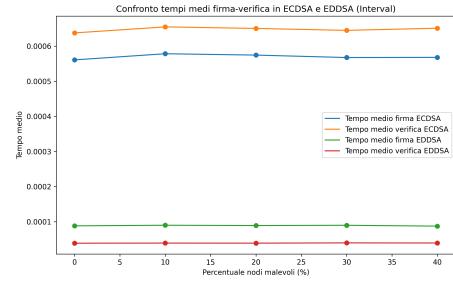


Figure 35: Tempo di firma-verifica (60 s)

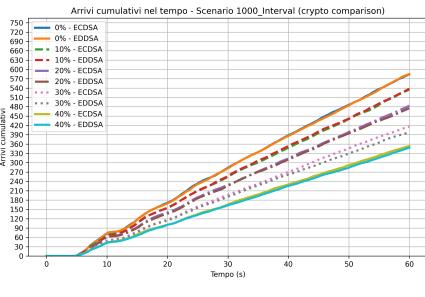


Figure 36: Arrivi cumulativi nel tempo (60 s)

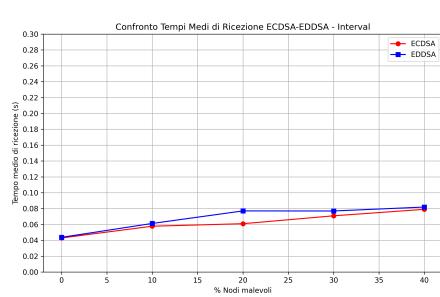


Figure 37: Reception Time (60 s)

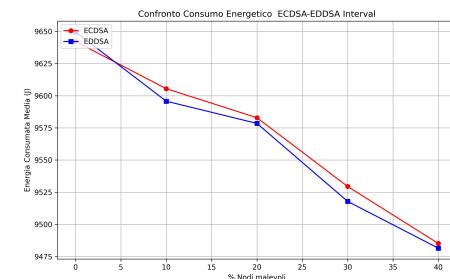


Figure 38: Consumo di energia (60 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** EDDSA si dimostra più veloce rispetto a ECDSA,

riducendo il tempo necessario per la generazione e la verifica delle firme digitali.

- **Numero medio di hop:** EDDSA consente di raggiungere la destinazione con un numero di hop inferiore rispetto a ECDSA, suggerendo un percorso di instradamento più efficiente.
- **Consumo energetico:** EDDSA risulta più efficiente dal punto di vista energetico, riducendo il dispendio di energia rispetto a ECDSA.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono un numero di pacchetti consegnati simile, senza differenze significative.
- **Tempo medio di ricezione:** le due modalità di firma presentano valori comparabili, indicando che la scelta dell'algoritmo di firma non incide significativamente sulla latenza complessiva della rete.

L'analisi evidenzia dunque che **EDDSA** offre vantaggi in termini di velocità di firma, efficienza energetica e ottimizzazione del percorso di routing, senza impattare negativamente sulla quantità di pacchetti consegnati o sui tempi medi di ricezione. Questi risultati suggeriscono che, in scenari con risorse limitate e necessità di basse latenze, EDDSA possa rappresentare una scelta preferibile rispetto a ECDSA.

#### 5.12.4 Confronto fra ECDSA e EDDSA 120 secondi

Estendendo il tempo di simulazione a 120 secondi, si osserva che le differenze tra **ECDSA** ed **EDDSA** tendono a ridursi, portando a prestazioni pressoché equivalenti nelle metriche considerate.

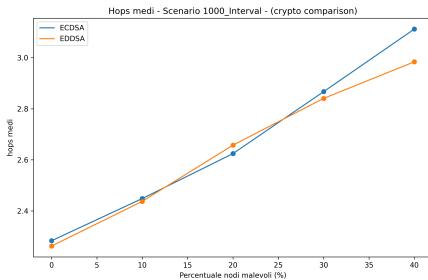


Figure 39: Mean Hops (120 s)

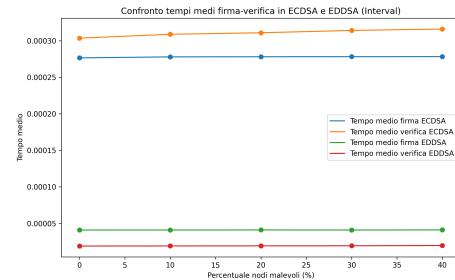


Figure 40: Tempo di firma-verifica (120 s)

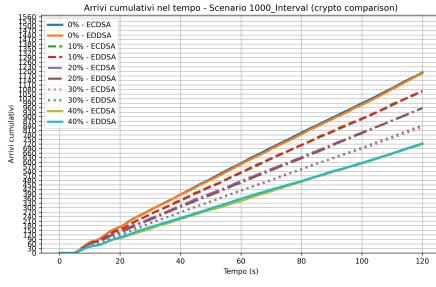


Figure 41: Arrivi cumulativi nel tempo (120 s)

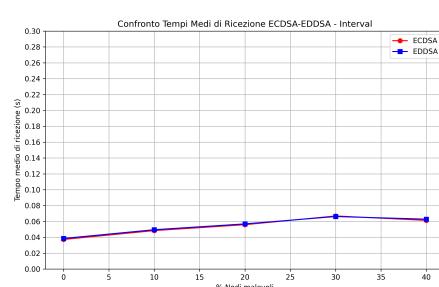


Figure 42: Reception Time (120 s)

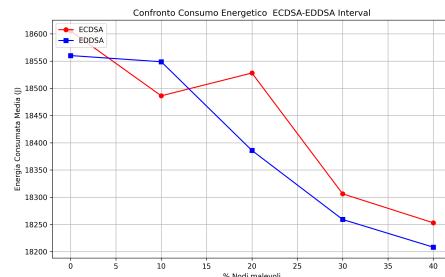


Figure 43: Consumo di energia (120 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** i due algoritmi presentano tempi di firma e verifica analoghi, senza variazioni significative rispetto alle simulazioni a 60 secondi.

- **Consumo energetico:** il dispendio energetico tra ECDSA ed EDDSA risulta pressoché identico alla modalità a 60s.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono una quantità di pacchetti ricevuti molto simile, senza evidenti differenze in termini di affidabilità della trasmissione.
- **Tempo medio di ricezione:** i tempi medi di consegna dei pacchetti rimangono invariati tra ECDSA ed EDDSA, confermando che l'algoritmo di firma non influenza significativamente sulla latenza complessiva della rete.
- **Numero medio di hop:** anche il numero di hop necessari per raggiungere la destinazione si mantiene pressoché identico tra le due configurazioni, senza variazioni degne di nota.

Questi risultati suggeriscono che, con tempi di simulazione più lunghi, le differenze tra ECDSA ed EDDSA si appiattiscono, rendendo entrambe le soluzioni ugualmente valide dal punto di vista delle prestazioni complessive del protocollo.

## 5.13 Conclusioni

Il sistema implementato integra la certificazione delle posizioni nei beacon con la verifica delle firme digitali, garantendo l'autenticità dei dati trasmessi e un monitoraggio dinamico della fiducia nei nodi della rete per contrastare l'operazione di falsificazione della posizione nei messaggi di GPSR **Beacon**. Di seguito si evidenziano i principali vantaggi della strategia:

1. **Integrità dei Dati:** L'impiego della firma digitale garantisce che le informazioni relative alla posizione siano autentiche, assicurando un elevato livello di sicurezza nelle comunicazioni.
2. **Affidabilità e Stabilità:** Il meccanismo di gestione della fiducia monitora costantemente il comportamento dei nodi, penalizzando quelli sospetti e rafforzando l'affidabilità di quelli corretti. L'adozione del meccanismo di smoothing previene variazioni brusche, contribuendo a mantenere il sistema stabile anche in presenza di anomalie.
3. **Resilienza:** Il sistema è progettato per identificare tempestivamente comportamenti fraudolenti e isolando i nodi malevoli. Questo approccio rende la rete più resiliente a potenziali attacchi di falsificazione della posizione.

**In sintesi**, l'adozione di posizioni certificate e la verifica delle firme digitali rappresentano una strategia efficace per garantire l'integrità e l'affidabilità dei dati riguardanti le posizioni dei nodi. Questo approccio si fonda su un lavoro sinergico tra le stazioni, che si occupano di firmare le posizioni, e i nodi, che scambiandosi messaggi di GPSR **Beacon** contententi la firma, ne verificano la validità in base alle informazioni riportate.

## 6 Scenario d'attacco 2: falsificazione della posizione agli altri nodi ed alle stazioni

### 6.1 Introduzione

Il protocollo GPSR (Greedy Perimeter Stateless Routing) basa il suo funzionamento sull'uso di informazioni geografiche per l'instradamento dei pacchetti in reti wireless. Questo approccio, sebbene efficiente in scenari ideali, diventa vulnerabile quando nodi malevoli alterano deliberatamente i dati di posizione. L'attacco analizzato sfrutta proprio questa debolezza attraverso un meccanismo sofisticato che contamina progressivamente le tabelle di routing della rete, compromettendone il corretto funzionamento senza richiedere risorse computazionali elevate.

### 6.2 Attacco

L'attacco prevede due fasi sequenziali.

Nella prima fase, i nodi compromessi generano posizioni geografiche false utilizzando tre strategie:

1. generano coordinate in quadranti diametralmente opposti rispetto alla posizione reale all'interno della mappa globale,
2. generano coordinate speculari rispetto al centro del quadrante locale,
3. generano semplicemente coordinate casuali nella mappa globale.

Questa variabilità rende difficile l'individuazione di eventuali pattern nella generazione delle posizioni false.

Nella seconda fase, le informazioni false vengono inoltrate sia ai nodi vicini che alle stazioni di riferimento. Ai vicini tramite i **beacon** periodici inviati in broadcast e alle stazioni tramite le **StationNotice**. Sia i vicini che le stazioni, ricevendo dati geografici alterati, aggiornano inconsapevolmente i loro registri posizionali, rispettivamente **neighborsPositionTable** e **quadNodesPositionTable**, con informazioni inconsistenti. Nella figura seguente *M1* falsa con la prima strategia, *M2* con la seconda e *M3* con la terza.

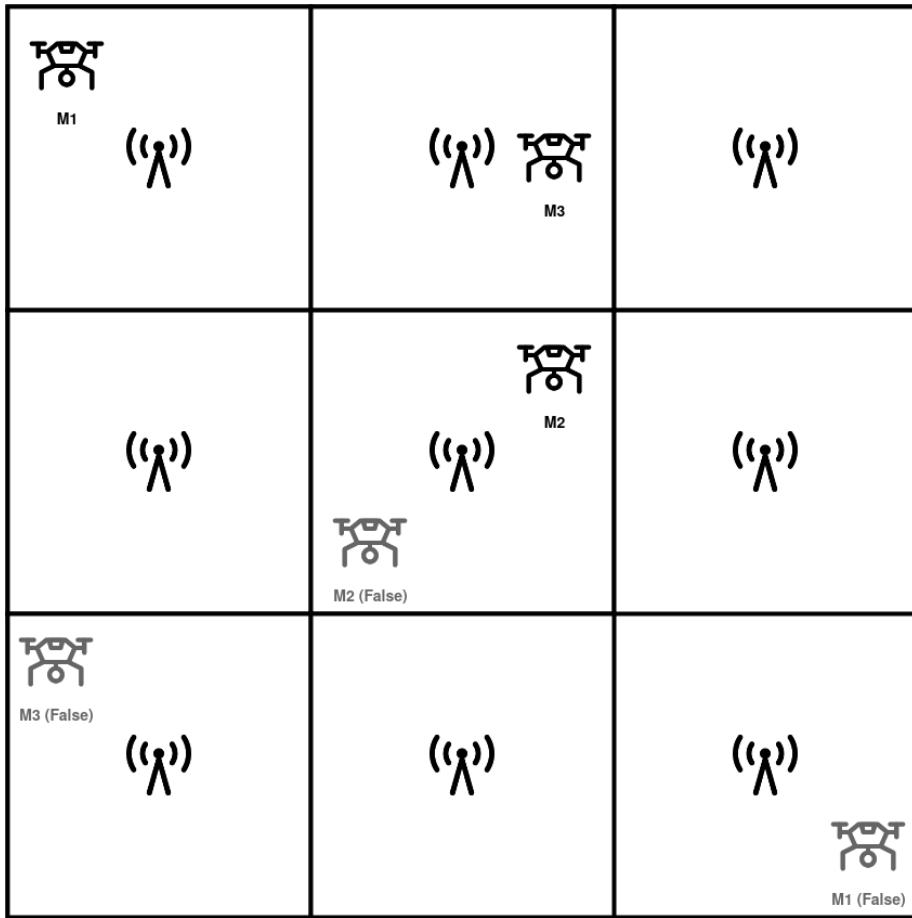


Figure 44: Esempio di generazione di posizioni false

### 6.3 Conseguenze

Le ripercussioni si manifestano su multipli livelli. A livello di routing, la fase greedy diventa inefficiente: i pacchetti vengono instradati verso "vicini più vicini" fantasma, aumentando esponenzialmente il numero di hop e attivando ripetutamente la costosa fase perimeter. Quest'ultima, a sua volta, genera dei loop, portando alla perdita di pacchetti.

## 6.4 Implementazione

---

**Algorithm 12** Generazione Posizione Falsa

---

**Require:** Posizione reale  $(x, y, z)$ , parametri  $n, m, z_{min}, z_{max}$

**Ensure:** Posizione falsa  $(x', y', z')$

```

1:  $\Delta \leftarrow n/m$ 
2:  $q_x \leftarrow \lfloor x/\Delta \rfloor$ 
3:  $q_y \leftarrow \lfloor y/\Delta \rfloor$ 
4:  $r \leftarrow \text{random}(0, 1)$ 
5: if  $r \leq 0.33$  then
6:   Strategia 1: Quadrante opposto
7:    $x' \sim U(0, \Delta) \cup U(n - \Delta, n)$ 
8:    $y' \sim U(0, \Delta) \cup U(n - \Delta, n)$ 
9: else if  $0.33 < r \leq 0.66$  then
10:  Strategia 2: Simmetria locale
11:   $c_x \leftarrow (q_x + 0.5)\Delta$ 
12:   $c_y \leftarrow (q_y + 0.5)\Delta$ 
13:   $x' \leftarrow 2c_x - x$ 
14:   $y' \leftarrow 2c_y - y$ 
15: else
16:   Strategia 3: Casuale
17:    $x' \sim U(0, n)$ 
18:    $y' \sim U(0, n)$ 
19: end if
20:  $z' \sim U(z_{min}, z_{max})$ 
21: return  $(x', y', z')$ 

```

---



---

**Algorithm 13** Creazione Beacon

---

```

1: address  $\leftarrow$  indirizzo nodo
2: realPos  $\leftarrow$  GETREALPOSITION
3: if isMalicious then
4:   pos  $\leftarrow$  GENERATEFALSEPOSITION(realPos)
5: else
6:   pos  $\leftarrow$  realPos
7: end if
8: return Beacon(address, pos)

```

---

---

**Algorithm 14** Creazione StationNotice

---

1: **Class Variables:**     $currentStation$ : indice stazione corrente2: **Input:**  $deregisterFlag$ 3:  $address \leftarrow$  indirizzo nodo4:  $realPos \leftarrow$  GETREALPOSITION5: **if**  $isMalicious$  **then**6:      $pos \leftarrow$  GENERATEFALSEPOSITION( $realPos$ )7: **else**8:      $pos \leftarrow realPos$ 9: **end if**10: **return** StationNotice( $address$ ,  $pos$ ,  $deregisterFlag$ )

---

## 6.5 Valutazioni delle performance

Le simulazioni in OMNeT++ sono state condotte in un'area di  $1000 \times 1000$  metri con un totale di 100 nodi. I nodi inviano periodicamente le proprie coordinate alle stazioni, che le memorizzano e le inoltrano agli altri nodi su richiesta. Alcune di questi nodi possono essere compromessi e falsificare le informazioni sulla posizione, alterando i dati diffusi nella rete. L'analisi si concentra sull'impatto di questa manipolazione sulle prestazioni del protocollo, valutando metriche come il tempo di consegna dei pacchetti, l'overhead del routing e il tempo impiegato nella perimeter search, per comprendere come la diffusione di informazioni errate possa degradare l'efficienza complessiva del sistema.

I parametri di simulazione adottati sono riportati nella Tabella 6.

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di nodi malevoli	0
Numero massimo di nodi malevoli	40
Tempo di simulazione	60/120 s
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 6: Parametri della simulazione

### 6.5.1 60 secondi

Tutti i grafici mostrati in questa sezione sono relativi alla modalità di invio *Interval*, che simula in maniera più realistica il comportamento di una rete wireless.

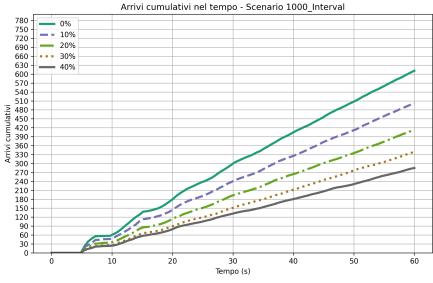


Figure 45: Arrivi cumulativi nel tempo (60 s)

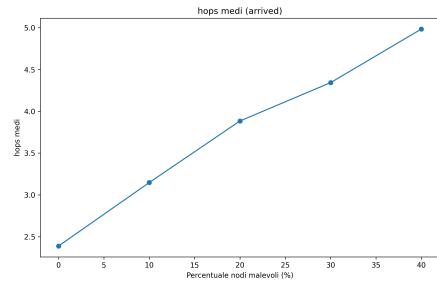


Figure 46: Mean Hops (60 s)

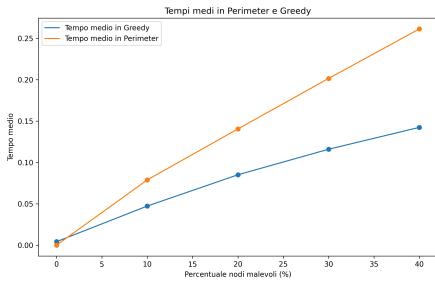


Figure 47: Per Greedy Time (60 s)

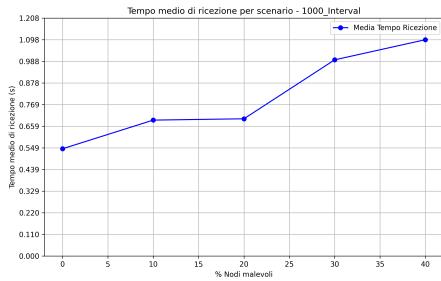


Figure 48: Reception Time (60 s)

E' possibile notare, come previsto, che all'aumento del numero di nodi malevoli, le prestazioni degradano. Il numero di messaggi arrivati a destinazione decresce mentre il numero di hops che questi impiegano per arrivare a destinazione aumenta; contestualmente, il rapporto tra perimeter time e greedy time aumenta così come il tempo medio che intercorre tra l'invio e la ricezione dei pacchetti, evidenziando la difficoltà del protocollo nel cercare path ottimali per il routing dei pacchetti.

### 6.5.2 120 secondi

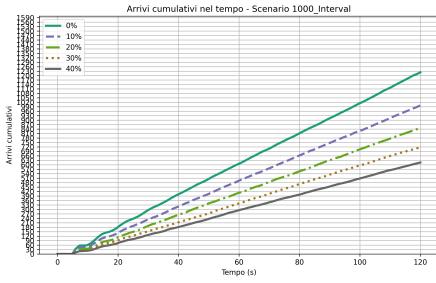


Figure 49: Arrivi cumulativi nel tempo (120 s)

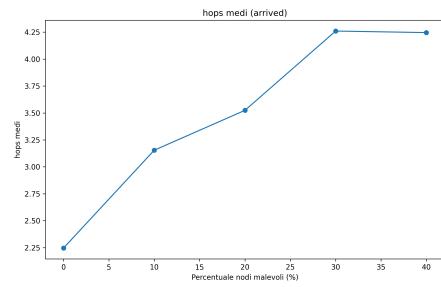


Figure 50: Mean Hops (120 s)

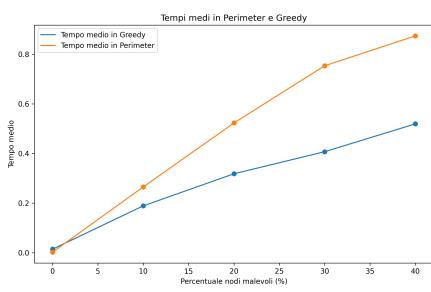


Figure 51: Per Greedy Time (120 s)

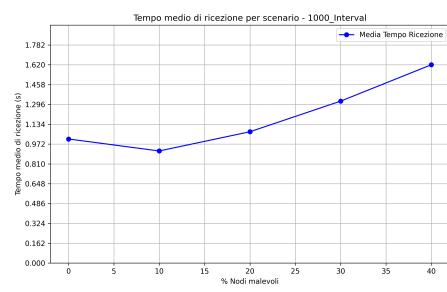


Figure 52: Reception Time (120 s)

Previdibilmente, il comportamento della rete è del tutto simile a quello visto nella sezione precedente.

## 7 Mitigazione per falsificazione della posizione agli altri nodi ed alle stazioni con triangolazione (Scenario d'attacco 2)

### 7.1 Meccanismo di mitigazione

Le fasi principali sono:

1. **Invio della stationNotice:** Ogni nodo invia periodicamente un messaggio `stationNotice` contenente:
  - Il proprio indirizzo.
  - La posizione dichiarata.
  - La lista dei vicini, in cui per ogni vicino vengono inclusi:
    - Il valore RSS (Received Signal Strength).
    - Un timestamp dell'ultimo contatto.
    - Un flag booleano che indica se la distanza stimata dall'RSS è coerente con quella calcolata dalla posizione ricevuta tramite beacon da parte del nodo. In generale viene fissata una soglia, se la differenza fra queste due distanze è al di sotto della soglia il flag viene impostato a *true*, *false* altrimenti.
2. **Elaborazione dei dati da parte della stazione:** Le stazioni ricevono il messaggio e incrociano i dati provenienti da più nodi per:
  - **Triangolazione:** Stimare la posizione reale del nodo verificando la coerenza tra le informazioni ricevute da diversi partecipanti.
  - **Calcolo del livello di trustness:** Assegnare al nodo un punteggio di fiducia basato sulla coerenza dei dati (ad es., validità degli RSS, correttezza dei timestamp e flag) e sulla cronologia della fiducia acquisita.
3. **Autenticazione e risposta:** Se il punteggio di trustness supera una soglia minima, la stazione:
  - Genera un **nonce**.
  - Firma digitalmente la coppia {posizione, nonce} usando la propria chiave privata.
  - Invia una risposta al nodo contenente la posizione confermata, il nonce e la firma.
4. **Diffusione e verifica tramite beacon:** Il nodo destinatario include le informazioni ricevute (posizione, firma e nonce) nei propri beacon, che vengono poi verificate dagli altri nodi mediante la scelta della chiave pubblica corretta, determinata in base al quadrante della posizione dichiarata.

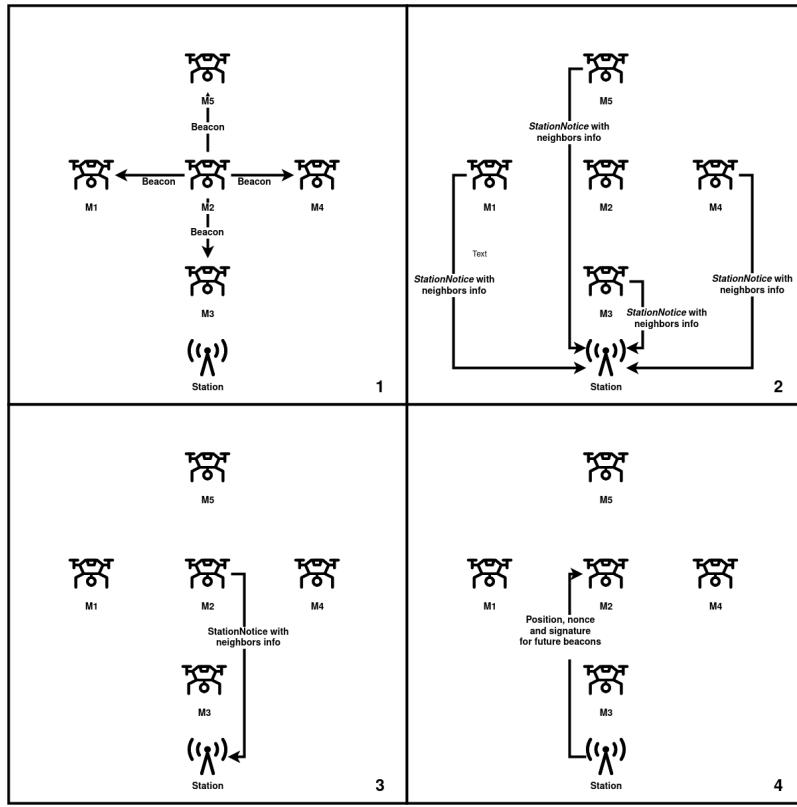


Figure 53: Esempio di mitigazione

Nella figura sopra viene presentato un esempio di come dovrebbe funzionare la mitigazione:

1. M2 manda i beacon ai suoi vicini, per semplicità si supponga che qui non si parli di nodi malevoli e che siano tutti legittimi e che quindi M2 abbia ottenuto in passato *nonce* e *signature* per i beacon che sta mandando.
2. Periodicamente M1, M3, M4 ed M5, mandano le *StationNotice* nelle quali ci saranno informazioni sui vicini e quindi anche informazioni su M2.
3. Anche M2 prima o poi dovrà mandare una *StationNotice*.
4. La stazione risponde con *nonce* e *signature* dato che può verificare il comportamento non malevolo di M2.

## 7.2 Elaborazione di una StationNotice da parte delle stazioni

La funzione `processStationNotice` implementa la logica di elaborazione dei messaggi `stationNotice` all'interno della stazione. Oltre alla semplice estrazione dei dati, l'algoritmo esegue diverse operazioni di validazione, incrocio dati e aggiornamento dinamico della fiducia.

## 7.3 Dettagli dell'Algoritmo

Di seguito si descrive in maniera più dettagliata il funzionamento dell'algoritmo:

### 1. Inizializzazione:

- *Estrazione:* Dal messaggio `stationNotice` vengono estratti:
  - Indirizzo del nodo (`address`).
  - La posizione dichiarata (`position`).
  - La lista dei vicini (`neighbors`).
- *Recupero trustness:* Viene recuperato l'ultimo valore di `trustness` registrato e associato al nodo.

### 2. Elaborazione dei dati dei vicini:

- Per ciascun vicino nella lista, vengono esaminate diverse informazioni:
  - *RSS e timestamp:* Verifica che il timestamp sia recente e che il valore RSS sia in linea con le aspettative.
  - *Flag di trustness:* Se il flag risulta falso, ciò indica una discrepanza tra la distanza stimata tramite RSS e quella calcolata dalla posizione dichiarata, quando il nodo ha ricevuto il beacon.
- In base all'analisi, il valore di trustness viene:
  - **Incrementato** se le informazioni sono coerenti.
  - **Decrementato** in caso di anomalie o dati incoerenti.

### 3. Triangolazione:

- L'algoritmo utilizza le informazioni provenienti dai vicini per eseguire una *triangolazione* e calcolare una posizione stimata basata sui dati incrociati.
- La posizione stimata viene confrontata con la posizione dichiarata (`position`). Se la differenza supera una certa soglia, ciò comporta una penalizzazione significativa del valore di trustness.

### 4. Calcolo finale del valore di trustness:

- Il valore calcolato viene combinato con l'ultimo valore di trustness registrato del nodo per avere una visione complessiva e dinamica dell'affidabilità.
- Questo aggiornamento include un meccanismo di *aging*.

### 5. Verifica della soglia ed eventuale invio della risposta:

- Se, al termine delle verifiche, il valore di trustness finale supera la soglia minima, si procede con l'autenticazione:
  - Viene generato un *nonce*.
  - La posizione dichiarata e il nonce vengono firmati digitalmente utilizzando la chiave privata della stazione.
  - Viene creato un messaggio di risposta contenente la posizione confermata, il nonce e la firma.
  - Il messaggio viene inviato al nodo che ha originariamente inviato la **stationNotice**.

---

**Algorithm 15** Elaborazione StationNotice da parte di una stazione

---

1: **Class Variables:**

$addressNeighborsMap$ : mappa che contiene per ogni indirizzo nel quadrante un lista di record dei vicini (ciascuno con RSS, timestamp e trustnessFlag)

$quadNodesPositionTable$ : mappa che contiene per ogni indirizzo nel quadrante la relativa posizione

2: **Input:** StationNotice contenente:

- **address**: indirizzo del nodo mittente
- **position**: posizione dichiarata dal nodo
- **neighbors**: lista di record dei vicini (ciascuno con RSS, timestamp e trustnessFlag)

3: **Output:** Se i controlli sono soddisfatti, invia un messaggio autenticato al nodo e aggiorna  $quadNodesPositionTable$

4:  $totalCloseNodes \leftarrow 0$

5:  $totalCloseNodesThatTrustAddress \leftarrow 0$

6:  $positionAndDistanceVector \leftarrow \{\}$

7: **for all**  $a \in addressNeighborsMap$  **do**

8:     **if**  $address \in addressNeighborsMap[a]$  **then**

9:          $ts \leftarrow GETTIMESTAMP(addressNeighborsMap[a], address)$

10:          $flag \leftarrow GETTRUSTNESSFLAG(addressNeighborsMap[a], address)$

11:          $rss \leftarrow GETRSS(addressNeighborsMap[a], address)$

12:         **if** ISVALID( $ts$ ) **then**

13:              $totalCloseNodes \leftarrow INCREASE$

14:              $closeNodePosition \leftarrow quadNodesPositionTable[a]$

15:              $closeNodeEstimatedDistance \leftarrow FROMRSSDISTANCE(rss)$

16:              $positionAndDistanceVector \leftarrow ADD(\{closeNodePosition,$

$closeNodeEstimatedDistance\})$

17:         **if**  $flag == true$  **then**

18:              $totalCloseNodesThatTrustAddress \leftarrow INCREASE$

19:         **end if**

20:         **end if**

21:     **end if**

22: **end for**

23:  $previousTrust \leftarrow GETPREVIOUSTRUST(address)$

24:  $estimatedPos \leftarrow TRIANGULATEPOSITION(positionAndDistanceVector)$

25:  $trustFromNeighbors \leftarrow GETTRUSTFROMNEIGHBORS(totalCloseNodes,$   
     $totalCloseNodesThatTrustAddress)$

26:  $trust \leftarrow CALCULATETRUST(previousTrust, estimatedPos, trustFromNeighbors)$

27:  $UPDATETRUST(address, trust)$

28: **if**  $trust \geq trustThreshold$  **then**

29:      $ADDTOPOSITIONTABLE(quadNodesPositionTable, address, position)$

30:      $nonce \leftarrow GENERATENONCE$

31:      $signature \leftarrow SIGN(position, nonce)$

32:      $SEND(StationNoticeResponse(position, signature, nonce))$

33: **end if**

---

## 7.4 Comportamento dei nodi malevoli

I nodi malevoli inviano informazioni false circa la loro posizione sia alle stazioni che agli altri nodi. In particolare, essi possono:

- a. **Inviare Posizioni False:** I nodi malevoli trasmettono un valore di `posNodo` non corrispondente alla loro reale posizione geografica, cercando di ingannare sia i vicini sia la stazione a terra.
- b. **Manipolare la Lista dei Vicini:** Nel messaggio `stationNotice` possono includere:
  - **Valori RSS falsificati:** Modificando il valore della potenza del segnale per alterare la stima della distanza.
  - **Flag di trustness errati:** Impostando il flag su `false` anche quando la distanza stimata è coerente, così da far sembrare malevoli dei nodi che si stanno comportando in maniera legittima.

## 7.5 Stima della distanza da *RSS*

---

**Algorithm 16** Calcolo della distanza da RSS

---

**Require:**  $P_r$ : potenza del segnale ricevuto in dBm

**Ensure:** Distanza stimata con rumore

- 1:  $P_t \leftarrow 3.01$
  - 2:  $f \leftarrow 2.4$
  - 3:  $c \leftarrow 3 \times 10^8$
  - 4: Calcola  $C \leftarrow 20 \cdot \log_{10}(f \times 10^9) + 20 \cdot \log_{10}\left(\frac{4\pi}{c}\right)$
  - 5: Calcola  $d_{\text{calc}} \leftarrow 10^{\frac{P_t - P_r - C}{20}}$
  - 6: Genera  $k_{\text{rumore}} \sim \mathcal{U}(1.01, 1.10)$
  - 7: **return**  $d_{\text{calc}} \times k_{\text{rumore}}$
- 

## 7.6 Calcolo del valore di *trustness*

La stazione calcola un valore di *trustness*  $T \in [T_{\min}, 1]$  per ogni nodo che transita nel quadrante, combinando:

- Verifiche dirette tramite **triangolazione** della posizione del nodo
- Feedback indiretti dai nodi vicini
- Andamento nel tempo del valore di trustness stesso

### 7.6.1 Parametri e variabili usate

Parametro	Descrizione	Effetto
$\alpha$	Fattore di smoothing	$\uparrow \alpha \Rightarrow$ più peso al passato
$T_{\min}$	Trustness minima	Evita oscillazioni molto forti
$w_{\text{vicini}}$	Peso reputazione	$\uparrow w_{\text{vicini}} \Rightarrow$ più affidamento sui feedback dei nodi
$\Delta_{\text{pen}}$	Penalità	Penalità in caso di mancanza di informazioni

Table 7: Tabella dei parametri

Variabile	Descrizione
$d_{\text{stimm}}$	Distanza tra la posizione dichiarata e la posizione stimata con triangolazione
$\mu_{\text{storico}}, \sigma_{\text{storico}}$	Media e deviazione standard tra le precedenti distanze stimate
$N_{\text{positivi}}, N_{\text{totale}}$	Feedback positivi e totali dai vicini
$T_{\text{prec}}$	Valore di trustness precedente

Table 8: Tabella delle variabili

### 7.6.2 Calcolo

Per il calcolo della *trustness* possono configurarsi due scenari:

1. La triangolazione è andata a buon fine ed è possibile ottenere la distanza dalla posizione dichiarata  $d_{\text{stimm}}$
2. La triangolazione non è andata a buon fine, quindi bisogna fare affidamento sui feedback dei nodi. Anche qui gli scenari possono essere due:
  - (a) Esistono dei feedback da consultare
  - (b) Non esistono dei feedback da consultare

Nel **Caso 1** il valore di *trustness* viene calcolato nel seguente modo:

$$\text{Soglia dinamica: } D_{\text{soglia}} = \mu_{\text{storico}} + \sigma_{\text{storico}} \quad (10)$$

$$\text{Trust dai vicini: } T_{\text{vicini}} = N_{\text{positivi}} / N_{\text{totale}} \quad (11)$$

$$\text{Trust dalla distanza: } T_{\text{distanza}} = e^{-d_{\text{stimm}} / D_{\text{soglia}}} \quad (12)$$

$$\text{Fusione: } T_{\text{new}} = w_{\text{vicini}} T_{\text{vicini}} + w_{\text{distanza}} T_{\text{distanza}} \quad (13)$$

$$\text{Aggiornamento: } T_{\text{agg}} = \alpha T_{\text{new}} + (1 - \alpha) T_{\text{prec}} \quad (14)$$

$$\text{Normalizzazione: } T = \max(T_{\min}, \min(1, T_{\text{agg}})) \quad (15)$$

Nel **Caso 2**, invece, il valore di *trustness* viene calcolato nel seguente modo:

$$\text{Aggiornamento: } T_{\text{agg}} = \begin{cases} \alpha T_{\text{prec}} + (1 - \alpha) T_{\text{vicini}}, & N_{\text{totale}} > 0 \\ \alpha T_{\text{prec}} + (1 - \alpha)(T_{\text{prec}} - \Delta_{\text{pen}}), & \text{altrimenti} \end{cases} \quad (16)$$

$$\text{Normalizzazione: } T = \max(T_{\min}, \min(1, T_{\text{agg}})) \quad (17)$$

### Spiegazione:

- La **soglia dinamica** (Eq. 1) adatta la sensibilità del sistema basandosi sulle precedenti distanze stimate
- Il **trustness dai vicini** (Eq. 2) assegna un valore di trustness proporzionale ai feedback positivi
- La **penalizzazione esponenziale** (Eq. 3) riduce la trustness per distanze superiori alla soglia
- La **fusione pesata** (Eq. 4) bilancia ciò che risulta dalla triangolazione e ciò che dichiarano i nodi, così da poter sfruttare sia l'uno che l'altro quando uno di questi sbaglia
- Lo **smoothing** (Eq. 5) evita oscillazioni brusche tenendo conto della storia passata
- Se sono disponibili **feedback dai vicini** (Eq. 7a):
  - Si sfrutta ciò che gli altri nodi pensano del nodo che ha mandato la `StationNotice`
  - Il valore di trustness evolve gradualmente in base a quello che pensano i vicini
- In **assenza di feedback** (Eq. 7b):
  - Si applica una penalità fissa  $\Delta_{\text{pen}}$
  - C'è una progressiva perdita di trustness se non si hanno informazioni
- La **normalizzazione** (Eq. 6/8) garantisce che il valore di trustness sia sempre tra  $T_{\min}$  e 1.

#### 7.6.3 Calcolo della soglia di *trustness* minima

Ogni stazione mantiene al proprio interno una mappa `trustness` la quale contiene le coppie:

$$\langle \text{address}, \text{trustnessValue} \rangle$$

In pratica per ogni nodo che transita nel suo quadrante, registra il suo valore di *trustness*. Per calcolare la soglia di *trustness* da confrontare si prende una percentuale del valore di *trustness* medio registrato in tabella.

$$\text{trustnessThreshold} = \begin{cases} \max \left( T_{\min}, \frac{1}{N} \sum_{i=1}^N T_i \times \beta \right) & \text{se } N > 0 \\ 1.0 & \text{altrimenti} \end{cases}$$

dove  $N = |\text{trustness}|$ ,  $T_i$  sono i valori nella mappa e  $\beta$  è un valore compreso tra 0 e 1.

## 7.7 Triangolazione

Gli algoritmi di triangolazione usati nella mitigazione sono sostanzialmente due, uno per quando si ha un numero di ancore pari a 3 ed uno per quando ce ne sono 4 o di più. Inoltre sono state integrate in ciascuno dei due delle tecniche di stabilizzazione del risultato per ottenere posizioni più coerenti possibile rispetto a quelle reali.

### 7.7.1 Algoritmo con 3 ancore

Questo metodo è usato per scenari con 3 ancore. Pur operando sostanzialmente in 2D, produce un risultato 3D fissando la coordinata verticale.

---

#### Algorithm 17 Triangolazione con 3 ancore

---

**Require:** Coordinate ancore  $(x_i, y_i, z_i)$ , distanze  $d_i$ ,  $i = 1, 2, 3$

**Ensure:** Coordinate stimate  $(x, y, z)$

1: Risolvi sistema lineare 2D:

$$\begin{cases} (x_2 - x_1)x + (y_2 - y_1)y = \frac{d_1^2 - d_2^2 + x_2^2 - x_1^2 + y_2^2 - y_1^2}{2} \\ (x_3 - x_1)x + (y_3 - y_1)y = \frac{d_1^2 - d_3^2 + x_3^2 - x_1^2 + y_3^2 - y_1^2}{2} \end{cases}$$

2: Fissa  $z = \frac{\min Z + \max Z}{2}$

3: Applica clamping:

$$x = \text{clamp}(x, \text{minX}, \text{maxX}), \quad y = \text{clamp}(y, \text{minY}, \text{maxY})$$

---

### 7.7.2 Algoritmo con $N \geq 4$ ancore

Fondamentalmente viene usato il metodo iterativo di Gauss-Newton, ottimizzando tutte e tre le coordinate simultaneamente, minimizzando l'errore quadratico medio.

---

#### Algorithm 18 Triangolazione con $N \geq 4$ ancore

---

**Require:**  $N \geq 4$  ancore  $(x_i, y_i, z_i)$ , distanze  $d_i$

**Ensure:** Coordinate ottimizzate  $(x, y, z)$

1: Inizializza  $\mathbf{p}_0 = (\frac{1}{N} \sum x_i, \frac{1}{N} \sum y_i, \frac{1}{N} \sum z_i)$

2: **while**  $\|\Delta\| > \epsilon$  **do**

3:     Calcola residui:

$$r_i = \sqrt{(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2} - d_i$$

4:     Calcola Jacobiano 3D:

$$J = \begin{bmatrix} \frac{x-x_1}{d_1} & \frac{y-y_1}{d_1} & \frac{z-z_1}{d_1} \\ \vdots & \vdots & \vdots \\ \frac{x-x_N}{d_N} & \frac{y-y_N}{d_N} & \frac{z-z_N}{d_N} \end{bmatrix}$$

5:     Risolvi  $(J^T J + \lambda I) \Delta = -J^T \mathbf{r}$

6:     Aggiorna:  $\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta$

7:     Applica clamping 3D

8:     Aggiorna  $\lambda$ :  $\lambda \leftarrow \begin{cases} \lambda/\beta & \text{se errore diminuisce} \\ \lambda \cdot \beta & \text{altrimenti} \end{cases}$

9: **end while**

10: **return**  $\mathbf{p}$

---

Nella fattispecie sono stati usati i seguenti parametri:

- $\epsilon = 10^{-6}$
- $\lambda_{\text{init}} = 10^{-4}$
- $\beta = 5$
- Numero massimo di iterazioni = 200

### 7.7.3 Tecniche di stabilizzazione dei risultati

Per migliorare robustezza e accuratezza in ambienti reali con rumore nelle misurazioni, sono state usate le seguenti tecniche di stabilizzazione:

- **Filtro sugli outlier:**

Dato

$$\text{Residuo}_i = |\|\mathbf{p} - \mathbf{a}_i\| - d_i| \quad (18)$$

Devono essere scartate le ancore con:

$$\text{Residuo}_i > \mu + 3\sigma \quad (19)$$

dove  $\mu$  e  $\sigma$  sono media e deviazione standard dei residui calcolati sul centroide iniziale.

- **Moving average filter:** per diminuire il rumore sui risultati viene fatta una media tra la posizione appena stimata per un nodo e le sue posizioni stimate nel passato, dando più peso a quelle recenti.

$$\mathbf{p}_{\text{filt}} = \frac{\mathbf{p}_{\text{new}} + \sum_{k=1}^M w_k \mathbf{p}_k}{1 + \sum_{k=1}^M w_k} \quad (20)$$

dove  $w_k = \frac{1}{1+(t_{\text{new}}-t_k)}$

- **Clamping:** dato che ogni stazione gestisce un dato quadrante, di cui, ovviamente conosce i limiti, se le stime hanno degli errori che porterebbero a soluzioni oltre il quadrante, queste vengono limitate.

$$\begin{aligned} x &= \max(x_{\min}, \min(x_{\max}, x)) \\ y &= \max(y_{\min}, \min(y_{\max}, y)) \\ z &= \max(z_{\min}, \min(z_{\max}, z)) \end{aligned} \quad (21)$$

- **Regolarizzazione adattiva:** Si è cercato di stabilizzare la matrice Hessiana in Gauss-Newton aggiungendo:

$$H_{\text{reg}} = H + \lambda I \quad (22)$$

Il parametro  $\lambda$  viene adattato dinamicamente durante le iterazioni.

In particolare il filtro sugli outlier è stato applicato ai dati prima della triangolazione, clamping e regolarizzazione durante le iterazioni e il *moving average filter* è stato applicato al risultato finale.

## 7.8 Valutazione delle performance

I parametri di simulazione adottati sono riportati nella Tabella 9

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di nodi malevoli	0
Numero massimo di nodi malevoli	40
Tempo di simulazione	60/120 s
Algoritmo di firma	ECDSA / EDDSA
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 9: Parametri della simulazione

### 7.8.1 60 secondi

Confrontiamo adesso i risultati ottenuti dalle simulazioni degli attacchi con quelli ottenuti dalle simulazioni con la mitigazione. Viene introdotta una nuova misura relativa al consumo energetico dei droni, al fine di valutare l'impatto energetico delle contromisure adottate. Per prima cosa visulizziamo i grafici con l'algoritmo di firma ECDSA.

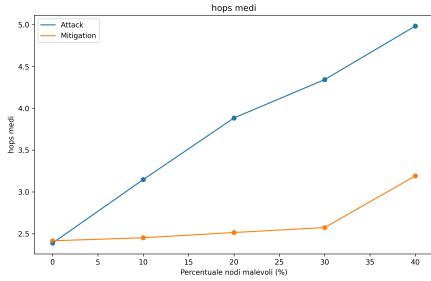


Figure 54: Mean Hops (60 s)

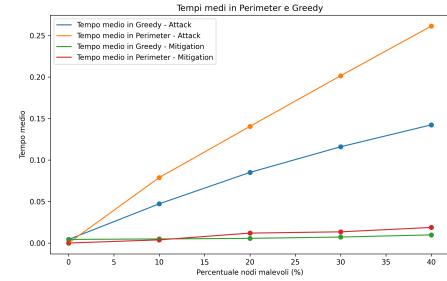


Figure 55: Per Greedy Time (60 s)

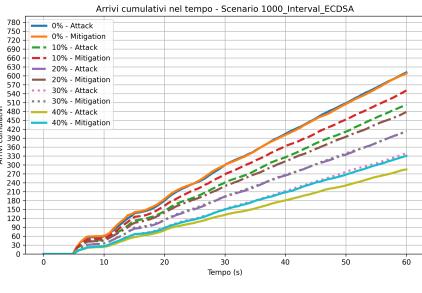


Figure 56: Arrivi cumulativi nel tempo (60 s)

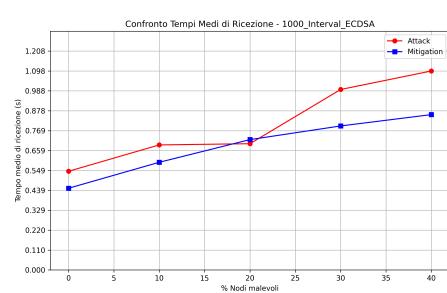


Figure 57: Reception Time (60 s)

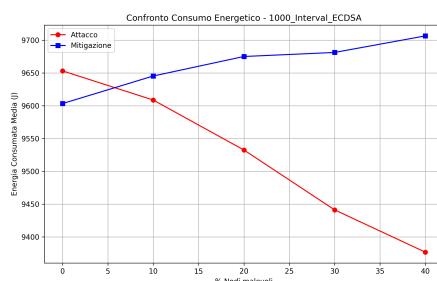


Figure 58: Consumo di energia (60 s)

Utilizzando l'algoritmo di firma EDDSA

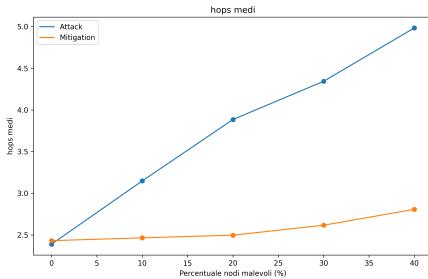


Figure 59: Mean Hops (60 s)

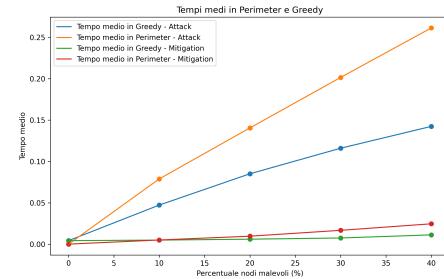


Figure 60: Per Greedy Time (60 s)

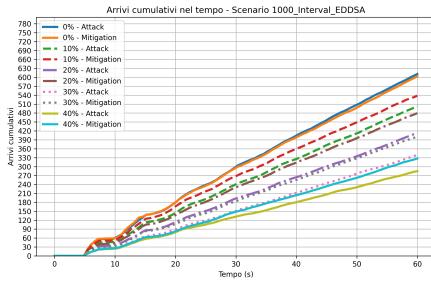


Figure 61: Arrivi cumulativi nel tempo (60 s)

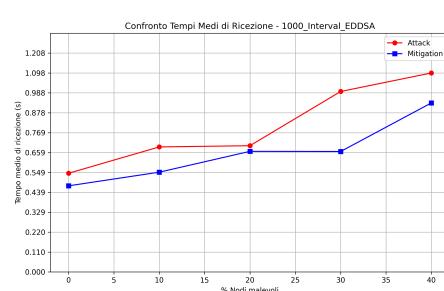


Figure 62: Reception Time (60 s)

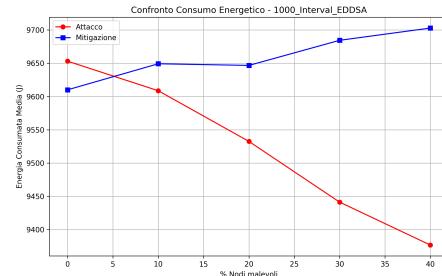


Figure 63: Consumo di energia (60 s)

Si può notare dai grafici come l'adozione delle mitigazioni migliori significativamente le prestazioni del protocollo:

- **Diminuzione del numero degli hop:** la strategia di avere posizioni validate e quindi inviare esclusivamente ai nodi con posizioni consistenti consente di mantenere una piena coerenza nel protocollo GPSR.
- **Maggiore numero di pacchetti ricevuti:** ignorando l'operato dei nodi malevoli, è possibile

garantire una piena coerenza nel resto dello scenario.

- **Diminuzione del tempo in modalità *perimeter*:** la piena coerenza del GPSR riduce la necessità di attivare il *perimeter forwarding*.

Questi benefici si osservano in entrambe le modalità di firma digitale, sia con **ECDSA** che con **EDDSA**, confermando l'efficacia del meccanismo di mitigazione indipendentemente dall'algoritmo crittografico adottato. In contrapposizione, si evince un aumento dei consumi dovuto alle attività di cifratura e decifratura dei nodi.

### 7.8.2 120 secondi

L'estensione del tempo di simulazione a 120 secondi conferma i benefici osservati con le mitigazioni. Anche in questo scenario, si registra un aumento del numero di pacchetti consegnati, una riduzione significativa del tempo medio di ricezione e del tempo trascorso in modalità *perimeter*. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano la stabilità e l'efficacia del protocollo di mitigazione su periodi di simulazione più lunghi.

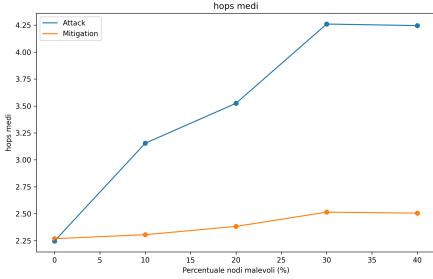


Figure 64: Mean Hops (120 s)

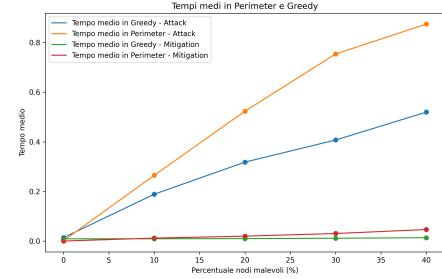


Figure 65: Per Greedy Time (120 s)

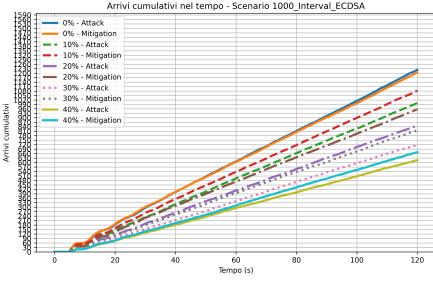


Figure 66: Arrivi cumulativi nel tempo (120 s)

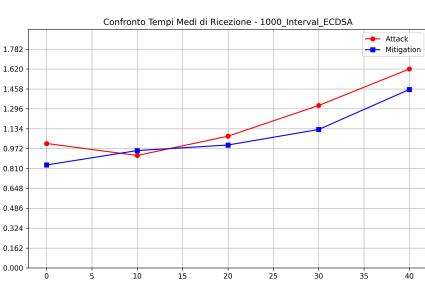


Figure 67: Reception Time (120 s)

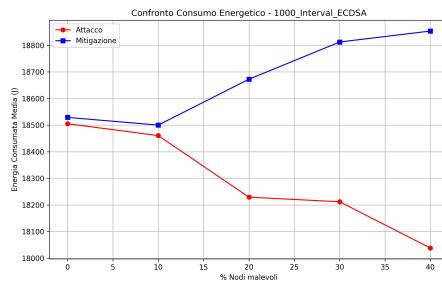


Figure 68: Consumo di energia (120 s)

Utilizzando l'algoritmo di firma **EDDSA**

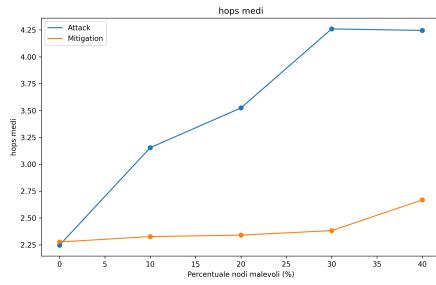


Figure 69: Mean Hops (120 s)

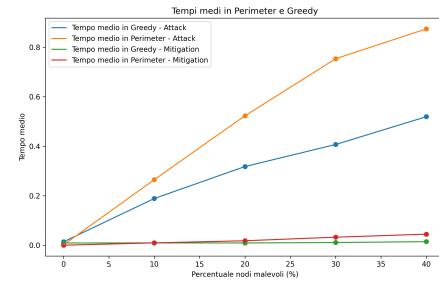


Figure 70: Per Greedy Time (120 s)

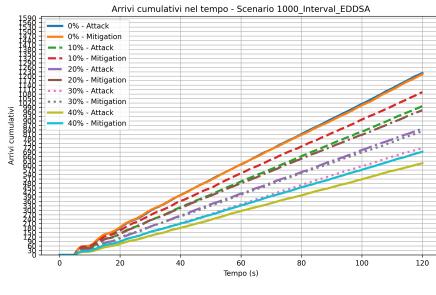


Figure 71: Arrivi cumulativi nel tempo (60 s)

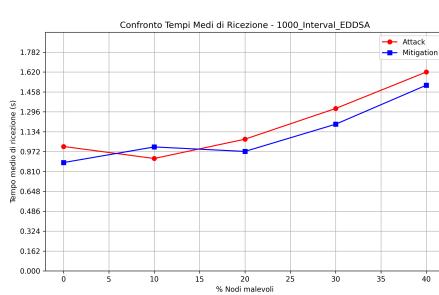


Figure 72: Reception Time (120 s)

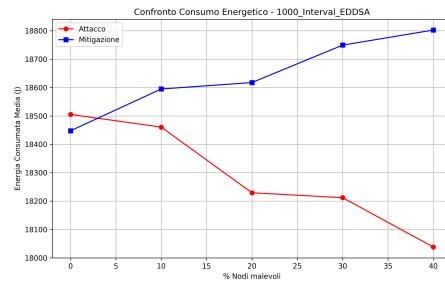


Figure 73: Consumo di energia (120 s)

Tuttavia, il consumo energetico risulta maggiore rispetto allo scenario senza mitigazioni, poiché i nodi eseguono operazioni aggiuntive per il controllo e l'esclusione dei nodi malevoli. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano l'efficacia del protocollo di mitigazione, pur con un costo energetico aggiuntivo.

### 7.8.3 Confronto fra ECDSA e EDDSA 60 secondi

Vediamo ora come differiscono i due algoritmi di firma digitale proposti, **ECDSA** ed **EDDSA**, valutando l'impatto che questi hanno sulle simulazioni avviate con i parametri precedenti, seguendo le stesse modalità di avvio, inserendo anche il *tempo medio di firma e verifica* per evidenziare meglio l'efficienza computazionale dei due algoritmi.

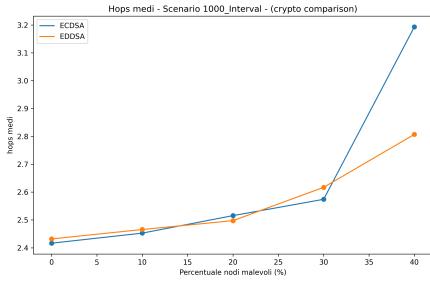


Figure 74: Mean Hops (60 s)

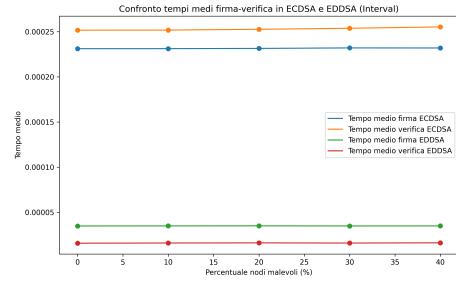


Figure 75: Tempo di firma-verifica (60 s)

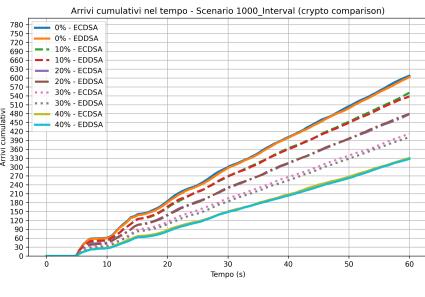


Figure 76: Arrivi cumulativi nel tempo (60 s)

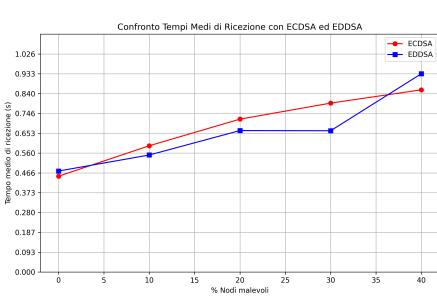


Figure 77: Reception Time (60 s)

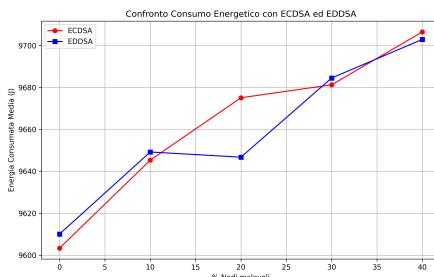


Figure 78: Consumo di energia (60 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** EDDSA si dimostra più veloce rispetto a ECDSA,

riducendo il tempo necessario per la generazione e la verifica delle firme digitali.

- **Numero medio di hop:** i due algoritmi si dimostrano entrambi abbastanza simili dal punto di vista di questa metrica, con EDDSA leggermente meglio all'aumentare del numero di malevoli, ECDSA meglio al diminuire.
- **Consumo energetico:** EDDSA risulta più efficiente dal punto di vista energetico.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono un numero di pacchetti consegnati simile, senza differenze significative.
- **Tempo medio di ricezione:** le due modalità di firma presentano valori comparabili, con dei valori leggermente a favore di EDDSA.

#### 7.8.4 Confronto fra ECDSA e EDDSA 120 secondi

Estendendo il tempo di simulazione a 120 secondi, si osserva che le differenze tra **ECDSA** ed **EDDSA** tendono a confermarsi su valori simili a quelli osservati in precedenza.

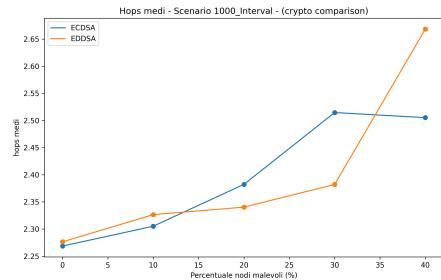


Figure 79: Mean Hops (120 s)

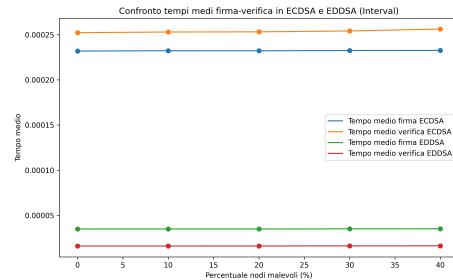


Figure 80: Tempo di firma-verifica (120 s)

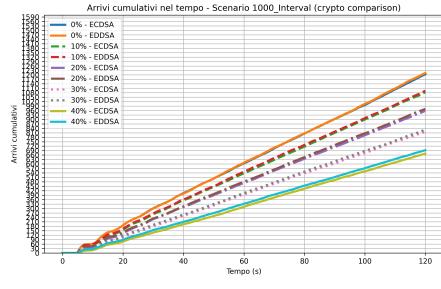


Figure 81: Arrivi cumulativi nel tempo (120 s)

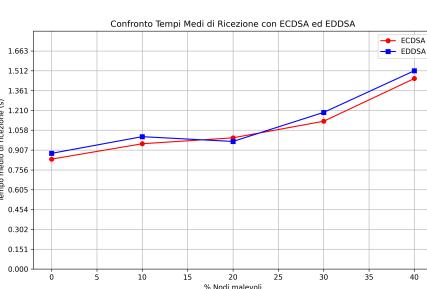


Figure 82: Reception Time (120 s)

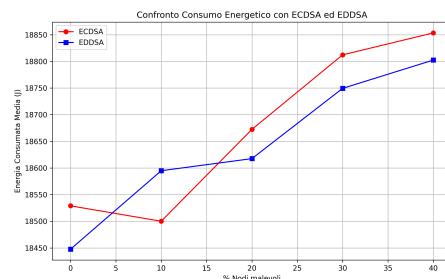


Figure 83: Consumo di energia (120 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** si riconferma l'osservazione fatta in precedenza con i due algoritmi che presentano tempi di firma e verifica analoghi.

- **Consumo energetico:** il dispendio energetico tra ECDSA ed EDDSA risulta confermarsi sui valori osservati nelle simulazioni precedenti.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono una quantità di pacchetti ricevuti molto simile, senza evidenti differenze in termini di affidabilità della trasmissione.
- **Tempo medio di ricezione:** i tempi medi di consegna dei pacchetti rimangono simili tra ECDSA ed EDDSA.
- **Numero medio di hop:** seppur la differenza sia lieve, in scenari più lunghi ECDSA risulta essere una soluzione migliori con un numero maggiore di nodi malevoli (40%), ma comunque peggiore in scenari intermedi (20%, 30%).

Questi risultati suggeriscono che, con tempi di simulazione più lunghi, le performance dei due algoritmi si mantengono stabili, confermando nella quasi totalità dei casi ciò che abbiamo osservato con le simulazioni più brevi.

## 7.9 Conclusioni

Il sistema implementato prevede un meccanismo di validazione incrociata e aggiornamento dinamico del valore di trustness, che permette alle stazioni di riconoscere e contrastare i tentativi di attacco. In sintesi:

- I nodi inviano periodicamente un messaggio `stationNotice` contenente la loro posizione e dettagli sui vicini.
- La stazione elabora questi messaggi effettuando controlli sui dati (inclusi RSS, timestamp, e flag di fiducia) e esegue una triangolazione per verificare la posizione dichiarata.
- Viene calcolato un nuovo livello di trustness, basato anche sui valori precedenti.
- Se il valore di trustness supera una soglia minima, la stazione autentica la posizione generando un nonce e firmando il messaggio, che viene inviato al nodo.
- I nodi malevoli tentano di manipolare questi parametri inviando informazioni false, ma il sistema incrociato e la verifica tramite triangolazione rendono più difficile il successo dell'attacco.

## 8 Mitigazione per falsificazione della posizione agli altri nodi ed alle stazioni senza triangolazione (Scenario d'attacco 2)

Il protocollo GPSR, come detto in precedenza, si basa sulle posizioni fisiche dei nodi per l'instradamento dei pacchetti, utilizzando una strategia *greedy* che inoltra i pacchetti al vicino più prossimo alla destinazione. Un attacco tipico in questo contesto, come visto, è lo **spoofing della posizione**, in cui un nodo malizioso dichiara una posizione falsa al fine di:

- Interrompere il corretto instradamento ;
- Isolare porzioni della rete.

Per contrastare tali attacchi viene proposto un meccanismo di mitigazione basato sul controllo di consistenza delle distanze e sulla gestione di una misura di reputazione (*trustness*). Tale meccanismo prevede l'interazione tra nodi e stazioni centrali che verificano e aggiornano dinamicamente il valore di fiducia associato a ogni nodo.

### 8.1 Definizioni e notazioni

Siano:

- $\mathcal{N}$  l'insieme dei nodi della rete.
- Per ogni nodo  $i \in \mathcal{N}$ ,  $P_i \in \mathbb{R}^3$  rappresenta la posizione dichiarata.
- La distanza tra due nodi  $i$  e  $j$  è definita come:

$$d_{ij}^{\text{declared}} = \|P_i - P_j\|.$$

- $d_{ij}^{\text{measured}}$  rappresenta la distanza misurata (o riportata dai vicini) tra  $i$  e  $j$ .
- $\delta$  è la tolleranza ammessa per considerare una distanza misurata come *consistente* rispetto a quella dichiarata.
- Il controllo di consistenza verifica che:

$$|d_{ij}^{\text{measured}} - d_{ij}^{\text{declared}}| \leq \delta.$$

- $T(i)$  denota la *reputazione* (o *trustness*) del nodo  $i$ .
- $\alpha \in (0, 1)$  è il coefficiente di aggiornamento (nel nostro caso,  $\alpha = 0.3$ ).
- PENALTY è una costante positiva che determina l'entità dell'incremento o della penalizzazione.
- $T_{\min}$  (o **notTrusted**) è la soglia al di sotto della quale un nodo viene considerato non affidabile e successivamente isolato.

## 8.2 Aggiornamento della trustness

Il meccanismo prevede che le stazioni centrali raccolgano, tramite messaggi `StationNotice`, le informazioni relative ai vicini e alle distanze misurate. Sia  $i$  il nodo in esame e  $j$  uno dei suoi vicini tale che  $T(j) > T_{\min}$ .

### 8.2.1 Stima della distanza

La distanza viene stimata mediante la differenza tra la potenza di invio del pacchetto e la potenza con la quale il pacchetto arriva al nodo  $n$ . Per la stima della distanza viene dunque adottato il seguente algoritmo, viene dunque aggiunto un fattore di distorsione per simulare un rumore.

---

#### Algorithm 19 Calcolo della distanza dalla potenza ricevuta (RSS)

---

```

1: function FROMRSSTODISTANCE(Pr)
2:    $Pt \leftarrow 3.01$                                  $\triangleright$  Imposta la potenza trasmessa
3:    $frequency \leftarrow 2.4$                            $\triangleright$  Imposta la frequenza (GHz)
4:    $c \leftarrow 3 \times 10^8$                           $\triangleright$  Velocità della luce (m/s)
5:    $C \leftarrow 20 \times \log_{10}(frequency \times 10^9) + 20 \times \log_{10}(\frac{4\pi}{c})$ 
6:    $noiseFactor \leftarrow 1.0 + (\text{random number between } 0 \text{ and } 1) \times 0.1$ 
7:    $distance \leftarrow 10^{\frac{(Pt - Pr - C)}{20}} \times noiseFactor$ 
8:   return  $distance$ 
9: end function
```

---

tale stima viene effettuata dal nodo  $n - esimo$ , se questa stima supera la soglia di tolleranza  $\delta$  allora alla stazione verrà inviato il quantitativo fittizio  $-1$ . Nel caso in cui tale stima venga effettuata dal nodo malevolo  $k$ -esimo, allora alla stazione verrà inviata una distanza falsificata. La stazione dunque procederà mediante la seguente fase di controllo:

#### Caso 1: Misurazione errata o incoerenza

Se la distanza misurata non è segnalata come errore ( $d_{ij}^{\text{measured}} = -1$ ) oppure se:

$$|d_{ij}^{\text{measured}} - d_{ij}^{\text{declared}}| > \delta,$$

allora il nodo  $i$  viene penalizzato aggiornando la sua reputazione secondo la formula:

$$T(i) \leftarrow (1 - \alpha) \cdot T(i) + \alpha \cdot (T(i) - \text{PENALTY}).$$

#### Caso 2: Misurazione coerente

Se invece si verifica che:

$$|d_{ij}^{\text{measured}} - d_{ij}^{\text{declared}}| \leq \delta,$$

il vicino  $j$  viene premiato, aggiornando la propria reputazione:

$$T(j) \leftarrow (1 - \alpha) \cdot T(j) + \alpha \cdot (T(j) + \text{PENALTY}).$$

## Aggiornamento in forma compatta

L'aggiornamento della reputazione, che adotta una forma di media mobile pesata, può essere scritto in modo compatto:

$$T_{\text{new}}(x) = (1 - \alpha) \cdot T(x) + \alpha \cdot (T(x) + \Delta),$$

dove:

$$\Delta = \begin{cases} -\text{PENALTY} & \text{se viene rilevata un'incoerenza (o errore),} \\ +\text{PENALTY} & \text{se la misurazione è coerente.} \end{cases}$$

## 8.3 Verifica basata sulle distanze

---

**Algorithm 20** Check Distance Consistency

---

```

function CHECKDISTANCECONSISTENCY(nodeAddress, reportedPosition, PENALTY)
    alpha  $\leftarrow$  0.3                                 $\triangleright$  Peso per il nuovo valore
    mapA  $\leftarrow$  {}                             $\triangleright$  Inizializza una mappa temporanea
    for indirizzo  $\in$  neighboursMap[nodeAddress] do
        if indirizzo  $\in$  neighboursMapDistance & trustness[indirizzo]  $>$  notTrusted then
            mapA  $\leftarrow$  neighboursMapDistance[indirizzo]  $\triangleright$  Recupera la mappa delle distanze
            distance  $\leftarrow$  mapA[nodeAddress]  $\triangleright$  Ottieni la distanza tra 'indirizzo' e 'nodeAddress'
            if distance  $=$  -1 then
                measurement  $\leftarrow$  trustness[nodeAddress]  $-$  PENALTY
                trustness[nodeAddress]  $\leftarrow$   $(1 - \alpha) \times \text{trustness[nodeAddress]} + \alpha \times$ 
measurement
            else
                measurement  $\leftarrow$  trustness[indirizzo]  $+ \text{PENALTY}$ 
                trustness[indirizzo]  $\leftarrow$   $(1 - \alpha) \times \text{trustness[indirizzo]} + \alpha \times \text{measurement}$ 
            end if
        end if
    end for
end function

```

---

Questo comportamento viene richiamato nel momento in cui un determinato nodo chiama 'stationNotice', quindi il meccanismo di controllo si centralizza all'interno della stazione, isolando quindi dalla rete il nodo che riporta un comportamento malizioso.

## 8.4 Gestione dei nodi maliziosi

Il meccanismo di mitigazione prevede inoltre che, se per un nodo  $i$  si verifica:

$$T(i) < T_{\min},$$

allora:

- La posizione  $P_i$  viene rimossa dalla tabella (ad es. `quadNodesPositionTable.removePosition`).
- Il nodo  $i$  viene classificato come malizioso e inserito in una lista di nodi compromessi (ad es. `maliciousIndividuati`).
- I riferimenti nei dati dei vicini vengono aggiornati (es. `neighboursMap.erase`), isolando ulteriormente il nodo.
- non viene fornita la risposta al nodo malevolo, non permettendo di proseguire con la registrazione presso la stazione, isolandolo definitivamente dalla rete.

## 8.5 Valutazione delle performance

Di seguito i parametri usati per le simulazioni:

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di nodi malevoli	0
Numero massimo di nodi malevoli	40
Tempo di simulazione	60/120 s
Algoritmo di firma	ECDSA / EDDSA
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 10: Parametri della simulazione

### 8.5.1 60s

In questa sezione vengono confrontati i risultati ottenuti in presenza dell'attacco con quelli ottenuti applicando la strategia di mitigazione con le metriche già analizzate.

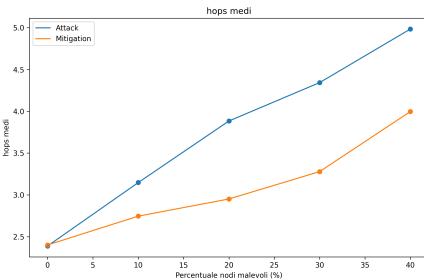


Figure 84: Mean Hops (60 s)

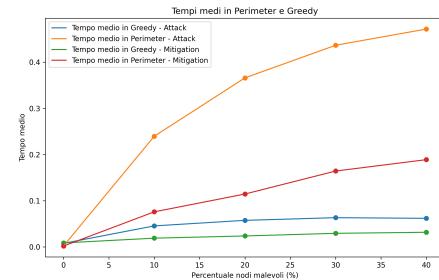


Figure 85: Per Greedy Time (60 s)

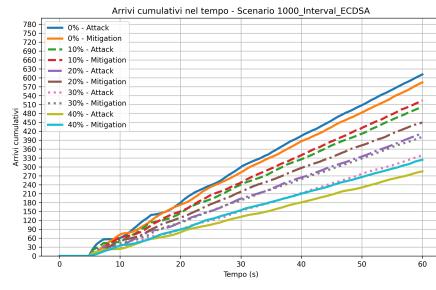


Figure 86: Arrivi cumulativi nel tempo (60 s)

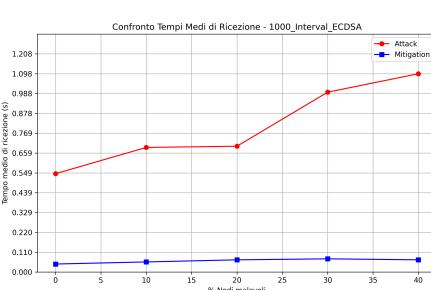


Figure 87: Reception Time (60 s)

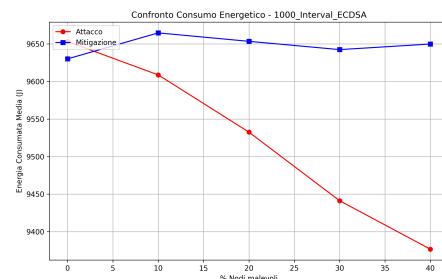


Figure 88: Consumo di energia (60 s)

Utilizzando l'algoritmo di firma EDDSA:

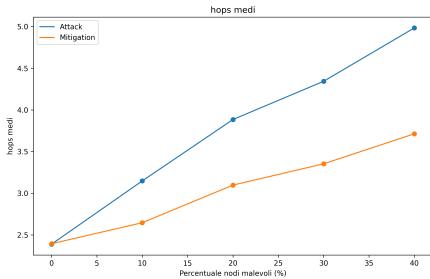


Figure 89: Mean Hops (60 s)

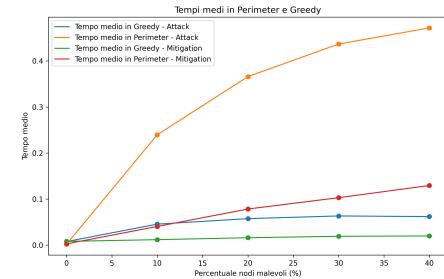


Figure 90: Per Greedy Time (60 s)

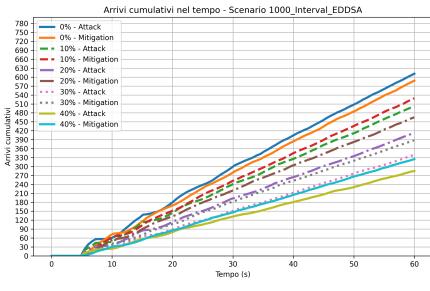


Figure 91: Arrivi cumulativi nel tempo (60 s)

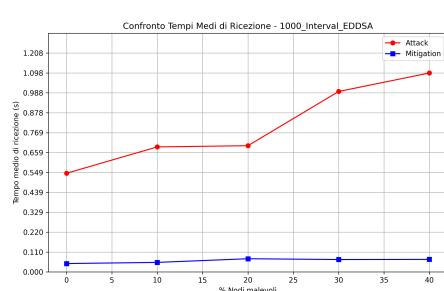


Figure 92: Reception Time (60 s)

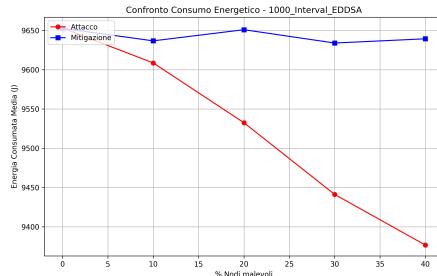


Figure 93: Consumo di energia (60 s)

L'analisi dei grafici evidenzia chiaramente come l'adozione delle mitigazioni migliori significativamente le prestazioni del protocollo:

- Maggiore numero di pacchetti ricevuti:** l'esclusione dei nodi malevoli dalle liste dei vicini consente di ripristinare percorsi di instradamento più affidabili, incrementando la probabilità di consegna dei pacchetti.
- Numero medio di hop:** La mitigazione consente di raggiungere la destinazione con un

numero di hop inferiore rispetto all'attacco.

- **Riduzione del tempo medio di ricezione:** la mitigazione riduce drasticamente il ritardo nella consegna dei pacchetti, permettendo una comunicazione più efficiente.
- **Diminuzione del tempo in modalità *perimeter*:** la corretta gestione delle liste dei vicini impedisce ai pacchetti di essere deviati su percorsi subottimali, riducendo la necessità di attivare il *perimeter forwarding*.

Questi benefici si osservano in entrambe le modalità di firma digitale, sia con **ECDSA** che con **EDDSA**, confermando l'efficacia del meccanismo di mitigazione indipendentemente dall'algoritmo crittografico adottato.

In contrapposizione, si evince un aumento dei consumi dovuto alle attività di cifratura e de-cifratura dei nodi che consumano molta energia

### 8.5.2 120s

L'estensione del tempo di simulazione a 120 secondi conferma i benefici osservati con le mitigazioni. Anche in questo scenario, si registra un aumento del numero di pacchetti consegnati, una riduzione significativa del tempo medio di ricezione, del tempo trascorso in modalità *perimeter* e del numero medio di hop per raggiungere la destinazione. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano la stabilità e l'efficacia del protocollo di mitigazione su periodi di simulazione più lunghi.

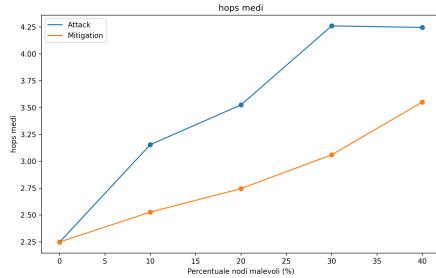


Figure 94: Mean Hops (120 s)

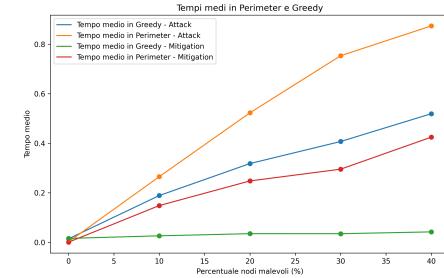


Figure 95: Per Greedy Time (120 s)

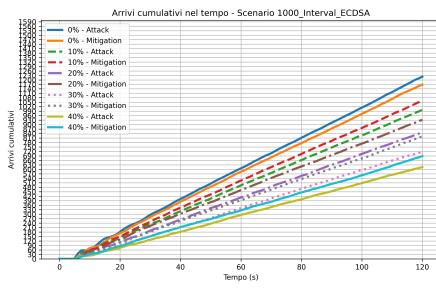


Figure 96: Arrivi cumulativi nel tempo (120 s)

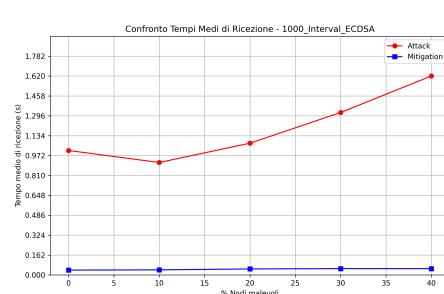


Figure 97: Reception Time (120 s)

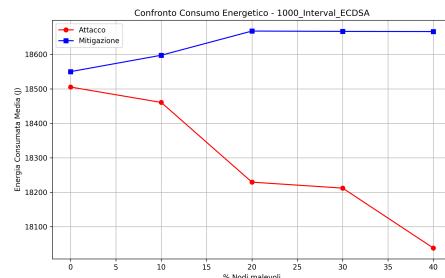


Figure 98: Consumo di energia (120 s)

Utilizzando l'algoritmo di firma EDDSA:

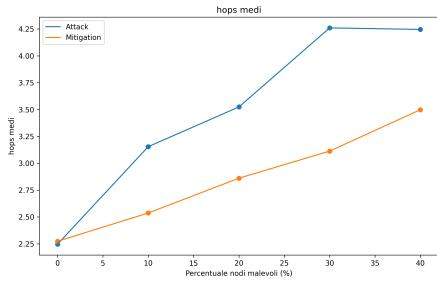


Figure 99: Mean Hops (120 s)

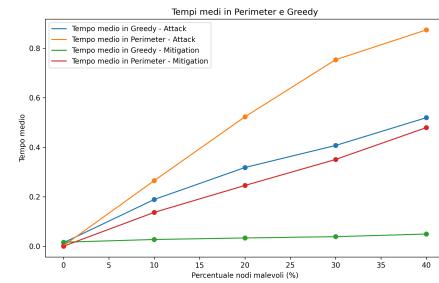


Figure 100: Per Greedy Time (120 s)

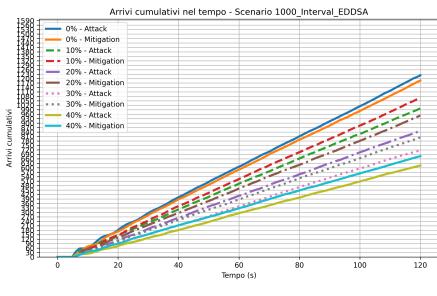


Figure 101: Arrivi cumulativi nel tempo (60 s)

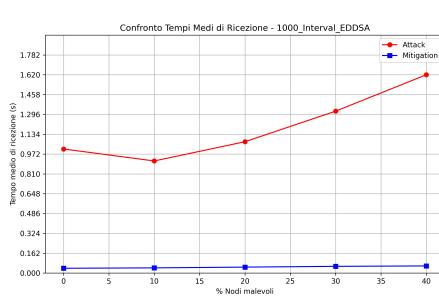


Figure 102: Reception Time (120 s)

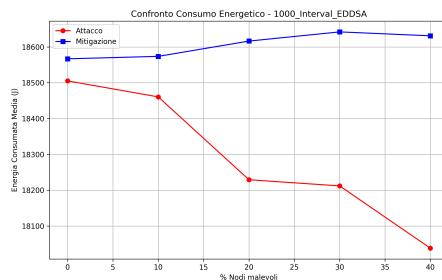


Figure 103: Consumo di energia (120 s)

Tuttavia, il consumo energetico risulta maggiore rispetto allo scenario senza mitigazioni, poiché i nodi eseguono operazioni aggiuntive per il controllo e l'esclusione dei nodi malevoli. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano l'efficacia del protocollo di mitigazione, pur con un costo energetico aggiuntivo.

### 8.5.3 Confronto fra ECDSA e EDDSA 60 secondi

In questa sezione viene analizzato il confronto tra le due modalità di firma digitale adottate, **ECDSA** ed **EDDSA**, valutandone l'impatto sulle metriche considerate. Oltre alle metriche già esaminate, viene introdotta la misura del *tempo medio di firma e verifica*, al fine di quantificare le differenze in termini di efficienza computazionale tra i due algoritmi.

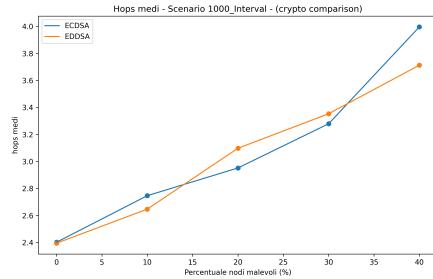


Figure 104: Mean Hops (60 s)

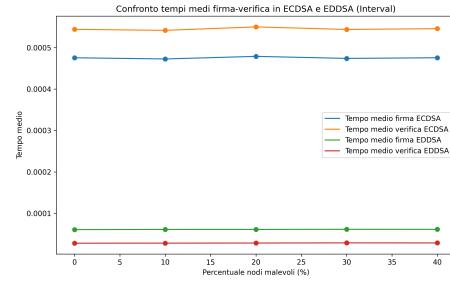


Figure 105: Tempo di firma-verifica (60 s)

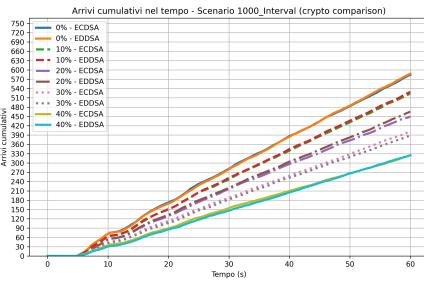


Figure 106: Arrivi cumulativi nel tempo (60 s)

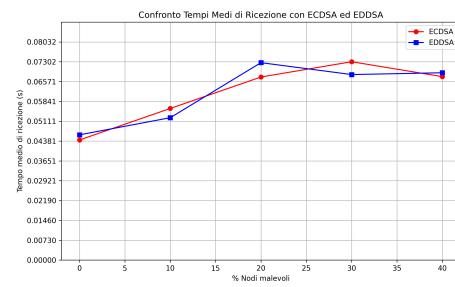


Figure 107: Reception Time (60 s)

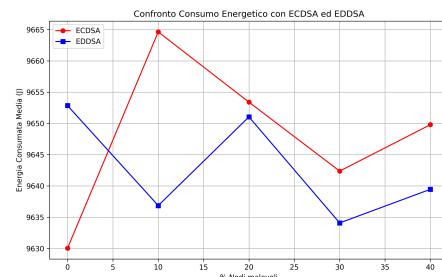


Figure 108: Consumo di energia (60 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** EDDSA si dimostra più veloce rispetto a ECDSA,

riducendo il tempo necessario per la generazione e la verifica delle firme digitali.

- **Consumo energetico:** EDDSA risulta più efficiente dal punto di vista energetico, riducendo il dispendio di energia rispetto a ECDSA.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono un numero di pacchetti consegnati simile, senza differenze significative.
- **Tempo medio di ricezione:** le due modalità di firma presentano valori comparabili, indicando che la scelta dell'algoritmo di firma non incide significativamente sulla latenza complessiva della rete.

L'analisi evidenzia dunque che **EDDSA** offre vantaggi in termini di velocità di firma, efficienza energetica, senza impattare negativamente sulla quantità di pacchetti consegnati o sui tempi medi di ricezione. Questi risultati suggeriscono che, in scenari con risorse limitate e necessità di basse latenze, EDDSA possa rappresentare una scelta preferibile rispetto a ECDSA.

#### 8.5.4 Confronto fra ECDSA e EDDSA 120 secondi

Estendendo il tempo di simulazione a 120 secondi, si osserva che le differenze tra **ECDSA** ed **EDDSA** tendono a ridursi, portando a prestazioni pressoché equivalenti nelle metriche considerate.

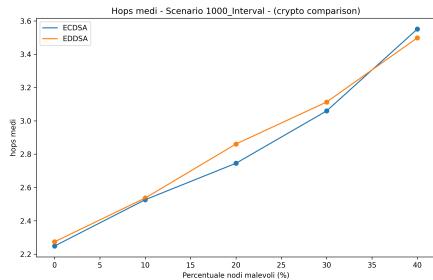


Figure 109: Mean Hops (120 s)

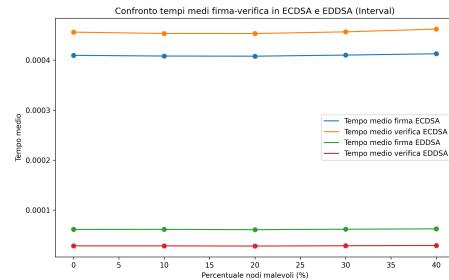


Figure 110: Tempo di firma-verifica (120 s)

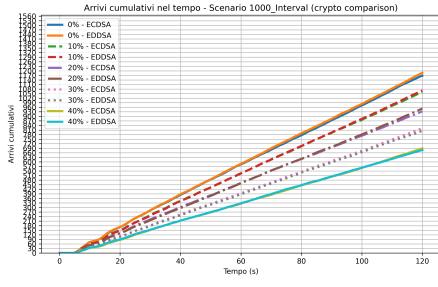


Figure 111: Arrivi cumulativi nel tempo (120 s)

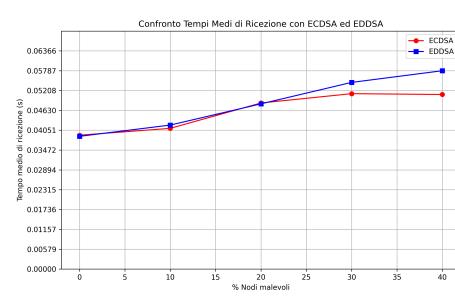


Figure 112: Reception Time (120 s)

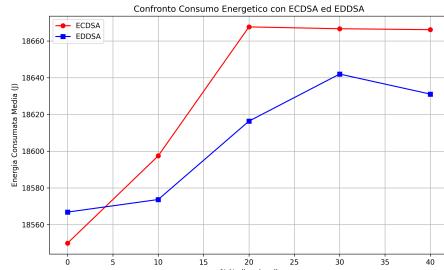


Figure 113: Consumo di energia (120 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** i due algoritmi presentano tempi di firma e verifica analoghi, senza variazioni significative rispetto alle simulazioni a 60 secondi.

- **Consumo energetico:** il dispendio energetico tra ECDSA ed EDDSA risulta pressoché identico alla versione a 60, con ECDSA che consuma di più.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono una quantità di pacchetti ricevuti molto simile, senza evidenti differenze in termini di affidabilità della trasmissione.
- **Tempo medio di ricezione:** i tempi medi di consegna dei pacchetti rimangono invariati tra ECDSA ed EDDSA, confermando che l'algoritmo di firma non influenza significativamente sulla latenza complessiva della rete.
- **Numero medio di hop:** anche il numero di hop necessari per raggiungere la destinazione si mantiene pressoché identico tra le due configurazioni, senza variazioni degne di nota.

Questi risultati suggeriscono che, con tempi di simulazione più lunghi, le differenze tra ECDSA ed EDDSA si appiattiscono, rendendo entrambe le soluzioni ugualmente valide dal punto di vista delle prestazioni complessive del protocollo.

## 8.6 Conclusioni

Il sistema proposto integra i seguenti elementi:

- **Verifica di consistenza:** Utilizzo dei dati locali (distanze misurate) per verificare la coerenza con le posizioni dichiarate.
- **Aggiornamento dinamico:** Utilizzo di un aggiornamento basato su media mobile pesata per adeguarsi a comportamenti variabili nel tempo.
- **Isolamento dei nodi malevoli:** Rimozione dei nodi la cui reputazione scende al di sotto di una soglia critica.

Questo dimostra come un controllo basato su metriche di affidabilità e verifiche incrociate possa mitigare attacchi di spoofing in protocolli di routing geografico come il GPSR.

## 9 Scenario d'attacco 3: stazioni inaffidabili o compromesse

### 9.1 Introduzione

L'architettura presa in considerazione in questo scenario è del tutto simile a quella dello scenario precedente, dove il funzionamento base del GPSR viene modificato introducendo delle **stazioni a terra** che hanno il compito di memorizzare una tabella di posizioni relative ad un quadrante di dimensione proporzionale a quella dell'intero scenario. In generale, uno scenario è costituito:

1. da un numero arbirtario di **droni**;
2. da un quadrato perfetto di **stazioni**.

I droni possono muoversi liberamente all'interno dello scneario, mentre le stazioni rimangono fisse al centro di ogni quadrante. Lo scenario è un quadrato di dimensione  $n \times n$  strutturato nel seguente modo:

- $n$  lunghezza,  $m$  stazioni per riga;
- $l = \frac{n}{m}$  distanza tra stazioni;
- $l \times l$  dimensione di un quadrante.

Il protocollo GPSR basa il suo funzionamento su una certa coerenza geografica necessaria per poter definire l'instradamento più corto. Quando questa coerenza viene a mancare, l'algoritmo diventa inevitabilmente meno efficiente. L'attacco dunque si basa sull'alterare le informazioni riguardanti le posizioni dei droni; in questo caso però non saranno i droni ad essere malevoli ed a falsificare la loro posizione ma le stazioni stesse, che, funzionando come un man-in-the-middle, alterano le informazioni recepite dai droni per inviarle agli altri droni (o alle altre stazioni) che le richiedono. Ogni drone comunica con la stazione del proprio quadrante attraverso i seguenti messaggi:

1. **StationNotice**: inviato da un drone alla stazione del proprio quadrante per registrare la propria posizione. Ogni stazione memorizzerà queste posizioni in una struttura dati, `PositionTable quadNodesPositionTable`.

StationNotice	Descrizione
L3Address address	Indirizzo del drone da registrare
Coord position	Coordinate del drone da registrare
bool flag	flag di registrazione/deregistrazione

*Strutture dati utilizzate: `quadNodesPositionTable` per memorizzare la coppia `<address, position>`*

2. **PositionRequest**: inviato da un drone alla stazione del proprio quadrante per richiedere la posizione di un altro drone.

PositionRequest	Descrizione
L3Address source	Indirizzo del drone richiedente
L3Address address	Indirizzo del drone destinazione

3. **S2SPositionRequest**: inviato da una stazione a un'altra stazione per richiedere la posizione di un drone. Quando una stazione riceve la richiesta per un drone di cui non ha informazioni, inoltrerà la richiesta ad un'altra stazione.

<b>S2SPositionRequest</b>	<b>Descrizione</b>
L3Address applicant	Indirizzo della stazione che ha inoltrato la richiesta
L3Address source	Indirizzo del drone che ha generato la richiesta
L3Address address	Indirizzo del drone destinazione

4. **PositionResponse**: inviato da una stazione a un drone in risposta a una **PositionRequest**.

<b>S2SPositionRequest</b>	<b>Descrizione</b>
bool setted	flag presenza informazioni drone destinazione
L3Address address	Indirizzo del drone destinazione
Coord pos	Coordinate del drone destinazione
simtime time	Tempo ultimo aggiornamento $<dest, pos>$

Strutture dati utilizzate: *quadNodesPositionTable* per recuperare *position* da *address*

5. **S2SPositionResponse**: inviato da una stazione a un'altra stazione in risposta a una **S2SPositionRequest**.

<b>S2SPositionResponse</b>	<b>Descrizione</b>
L3Address applicant	Indirizzo della stazione che ha inoltrato la richiesta
bool setted	flag presenza informazioni drone destinazione
L3Address address	Indirizzo del drone destinazione
Coord pos	Coordinate del drone destinazione
simtime time	Tempo ultimo aggiornamento $<dest, pos>$

Strutture dati utilizzate: *quadNodesPositionTable* per recuperare *position* da *address*

## 9.2 Logica dell'attacco

Concentriamoci sul comportamento delle stazioni malevoli. Il comportamento delle stazioni è dettato dal modo in cui processano le richieste dei droni o delle altre stazioni.

- **StationNotice**: quando una stazione malevola riceve un messaggio di questo tipo, si comporta come una stazione buona, memorizzando le informazioni sulla posizione ricevuta dal drone;

---

### Algorithm 21 Processamento di StationNotice - Attacco

---

```

1: Input: bool dereg, L3Address source, Coord position
2: if dereg then
3:   quadNodesPositionTable.remove(source)
4: else
5:   quadNodesPositionTable.add(source, position)
6: end if

```

---

- **PositionRequest:** in questo caso, il drone richiede la posizione di un altro drone; la stazione malevola risponde in ogni caso (sia se le informazioni sul drone siano presenti o meno) generando positionResponse; notare che la stazione malevola non verifica la presenza della posizione nella tabella apposita;

---

**Algorithm 22** Processamento di PositionRequest - Attacco

---

```

1: Input: L3Address destination
2: Output: Coord pos, bool setted
3: if quadNodesPositionTable.has(destination) OR isMalicious then
4:   if isMalicious then
5:     pos  $\leftarrow$  generateFalsePosition()
6:   else
7:     pos  $\leftarrow$  quadNodesPositionTable.get(destination)
8:   end if
9:   setted  $\leftarrow$  True
10: else
11:   setted  $\leftarrow$  False
12: end if
13: return (pos, setted)

```

---

- **S2SPositionRequest:** in questo caso, è la stazione a richiedere una posizione di un drone. La logica non varia rispetto al caso precedente.

---

**Algorithm 23** Processamento di S2SPositionRequest - Attacco

---

```

1: Input: L3Address applicant, L3Address destination
2: Output: Coord pos, bool setted
3: if quadNodesPositionTable.has(destination) OR isMalicious then
4:   if isMalicious then
5:     pos  $\leftarrow$  generateFalsePosition()
6:   else
7:     pos  $\leftarrow$  quadNodesPositionTable.get(destination)
8:   end if
9:   setted  $\leftarrow$  True
10: else
11:   setted  $\leftarrow$  False
12: end if
13: return (pos, setted, applicant)

```

---

Implementando in questo modo la logica dell'attacco, le stazioni malevoli rispondono *sempre* ad una richiesta con una posizione *non consistente*: in questo modo si minaccia l'efficienza del protocollo GPSR.

### 9.3 Generazione di posizioni false

Il metodo per generare posizione falsa si basa sullo scegliere con probabilità uniforme una delle seguenti strategie:

- generazione di una posizione falsa in modo speculare al centro della mappa (es.  $M1$ );
- generazione di una posizione falsa in modo speculare alla mia posizione (es.  $M2$ );
- generazione di una posizione falsa con coordinate casuali (es.  $M3$ ).

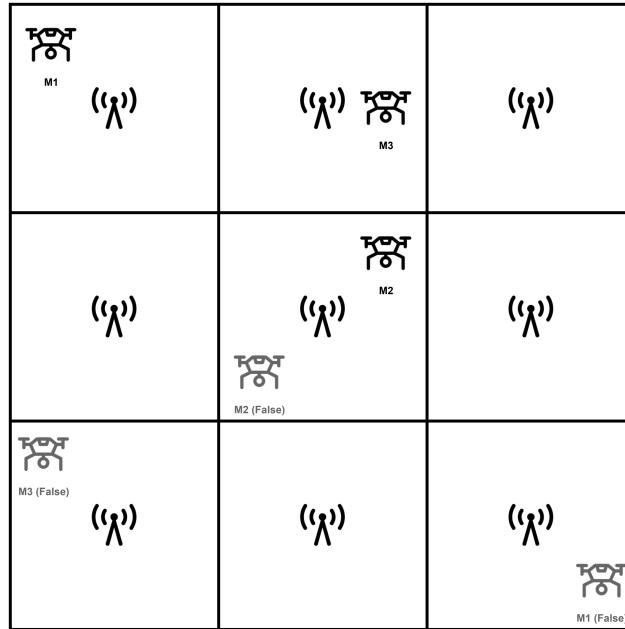


Figure 114: Generazione posizione falsa

---

#### Algorithm 24 Generazione di una posizione falsa

---

```

1: Output: Coord falsePos
2: prob  $\leftarrow$  uniform(0.0, 1.0)
3: if prob  $\leq$  0.33 then
4:   falsePos  $\leftarrow$  SimmetricaAllaMappa()
5: else if 0.33 < prob  $\leq$  0.66 then
6:   falsePos  $\leftarrow$  SimmetricaAlQuadrante()
7: else
8:   falsePos  $\leftarrow$  Random()
9: end if
10: return falsePos

```

---

**Nota:** Non c'è bisogno di basare la falsificazione sulla posizione reale del drone: rendendo l'approccio totalmente casuale o basato su informazioni banali (come ad esempio la posizione della stazione stessa) si implementa sia una logica computazionalmente più semplice, sia si isola di più l'informazione riguardo la posizione reale del drone.

#### 9.4 Valutazioni delle performance

Le simulazioni in OMNeT++ sono state condotte in un'area di  $1000 \times 1000$  metri con un totale di 100 nodi. I nodi inviano periodicamente le proprie coordinate alle stazioni, che le memorizzano e le inoltrano agli altri nodi su richiesta. Alcune di queste stazioni possono essere compromesse e falsificare le informazioni sulla posizione, alterando i dati diffusi nella rete. L'analisi si concentra sull'impatto di questa manipolazione sulle prestazioni del protocollo, valutando metriche come il tempo di consegna dei pacchetti, l'overhead del routing e il tempo impiegato nella perimeter search, per comprendere come la diffusione di informazioni errate possa degradare l'efficienza complessiva del sistema.

I parametri di simulazione adottati sono riportati nella Tabella 11.

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di stazioni malevole	0
Numero massimo di stazioni malevole	4
Tempo di simulazione	60/120 s
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 11: Parametri della simulazione

#### 9.4.1 60 secondi

Tutti i grafici mostrati in questa sezione sono relativi alla modalità di invio *Interval*, che simula in maniera più realistica il comportamento di una rete wireless.

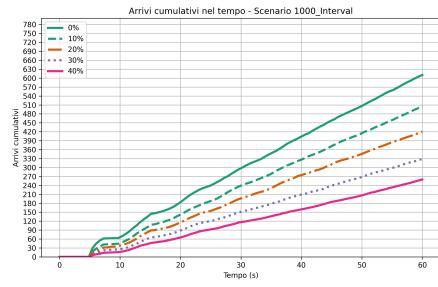


Figure 115: Arrivi cumulativi nel tempo (60 s)

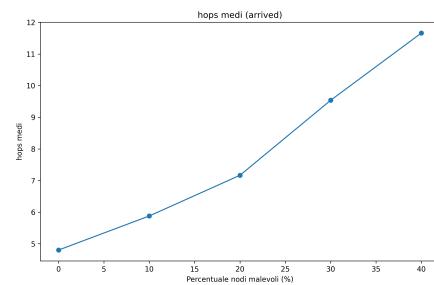


Figure 116: Mean Hops (60 s)

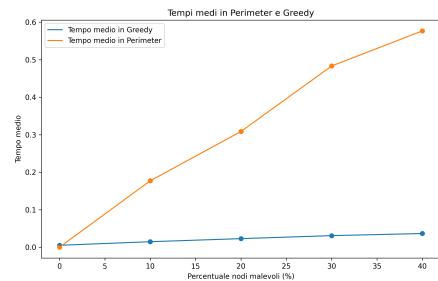


Figure 117: Per Greedy Time (60 s)

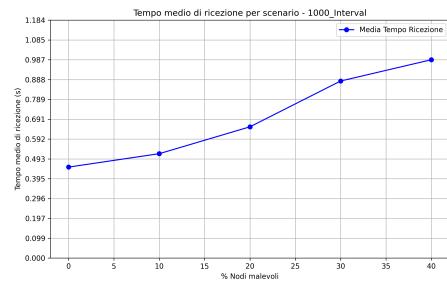


Figure 118: Reception Time (60 s)

E' possibile notare, come previsto, che all'aumento del numero di stazioni malevole, le prestazioni degradano. Il numero di messaggi arrivati a destinazione decresce mentre il numero di hops che questi impiegano per arrivare a destinazione aumenta; contestualmente, il rapporto tra perimeter time e greedy time aumenta così come il tempo medio che intercorre tra l'invio e la ricezione dei pacchetti, evidenziando la difficoltà del protocollo nel cercare path ottimali per il routing dei pacchetti.

#### 9.4.2 120 secondi

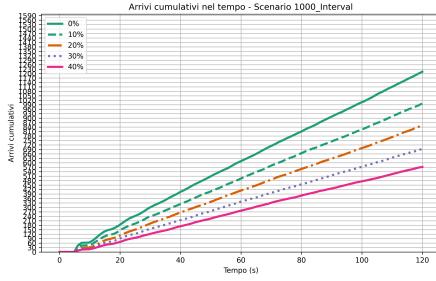


Figure 119: Arrivi cumulativi nel tempo (120 s)

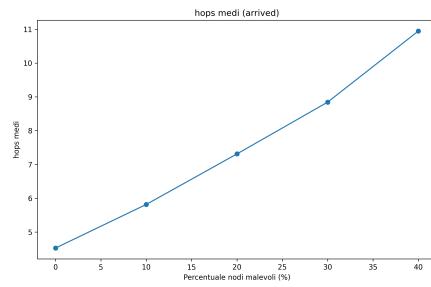


Figure 120: Mean Hops (120 s)

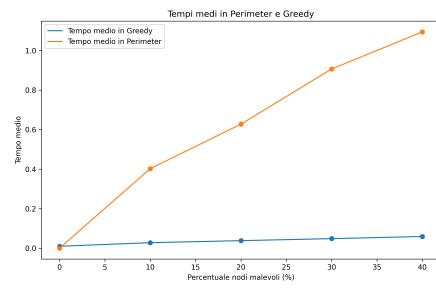


Figure 121: Per Greedy Time (120 s)

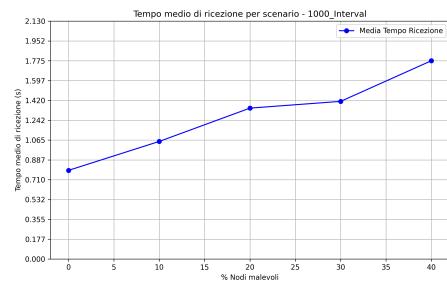


Figure 122: Reception Time (120 s)

Previdibilmente, il comportamento della rete è del tutto simile a quello visto nella sezione precedente.

## 9.5 Conseguenze

Le conseguenze risultano abbastanza evidenti: in questo caso, l'intero quadrante di una stazione malevola e tutte le altre stazioni che la consultano ricevono informazioni errate, rendendo il protocollo totalmente inefficiente quando l'instradamento ottimale secondo il GPSR richiederebbe il passaggio del pacchetto ad uno dei droni nel quadrante della stazione malevola. Questo porta ad un aumento del numero di hop medi affinché un pacchetto arrivi a destinazione ed un minor numero di pacchetti arrivati a destinazione.

## 10 Mitigazione per stazioni inaffidabili o compromesse (Scenario d'attacco 3)

La mitigazione consiste sostanzialmente in due fasi:

1. Identificazione delle stazioni malevole;
2. Limitazioni dei danni;

### 10.1 Identificazione

Per l'identificazione delle stazioni malevole, si è scelto di utilizzare un meccanismo di autenticazione a chiave asimmetrica. Tutti gli attori possiedono una coppia di chiavi, utilizzano la propria chiave privata per firmare e quella pubblica degli altri attori per verifarne la firma. Inoltre, si fa affidamento ad una struttura dati: `map<L3Address, tuple<Coord, string, uint64_t, simtime_t>> quadNodesSignatureMap` che permette alle stazioni di memorizzare la firma relativa alla posizione di un drone. I messaggi coinvolti in questo meccanismo sono tutti quelli scambiati dalla rete. In particolare:

- **StationNotice:** la struttura di questo messaggio varia includendo la firma del drone che ha generato il messaggio (e relativo nonce), la stazione provvederà a memorizzare la firma e la posizione nella nuova struttura dati precedentemente introdotta (con relativo nonce);

StationNotice	Descrizione
L3Address source	
Coord position	
bool deregister	
string <b>droneSignature</b>	firma del drone
uint64 <b>droneNonce</b>	nonce relativo alla firma

*Strutture dati utilizzate: quadNodesPositionTable per memorizzare la coppia <address, position>; quadNodesSignatureMap per memorizzare la tupla <signature, nonce, simtime>*

- **PositionResponse:** la struttura di questo messaggio varia includendo la firma della stazione che lo ha generato e la firma relativa alla posizione del drone considerato (con relativi nonce), recuperata da `quadNodesSignatureMap`;

PositionResponse	Descrizione
bool setted	
L3Address address	
Coord position	
simtime time	
string <b>stationSignature</b>	firma della stazione
uint64 <b>stationNonce</b>	nonce relativo alla firma della stazione
string <b>droneSignature</b>	firma del drone
uint64 <b>droneNonce</b>	nonce relativo alla firma del drone

*Strutture dati utilizzate:* `quadNodesPositionTable` per recuperare la `position` da `address`; `quadNodesSignatureMap` per recuperare la tupla `<signature, nonce, simtime>` da `address`

- **S2SPositionResponse:** la struttura di questo messaggio varia in maniera simile a Position-Response.

S2SPositionResponse	Descrizione
L3 Address applicant bool setted L3Address address Coord position simtime time	
string <b>stationSignature</b>	firma della stazione
uint64 <b>stationNonce</b>	nonce relativo alla firma della stazione
string <b>droneSignature</b>	firma del drone
uint64 <b>droneNonce</b>	nonce relativo alla firma del drone

*Strutture dati utilizzate:* `quadNodesPositionTable` per recuperare la `position` da `address`; `quadNodesSignatureMap` per recuperare la tupla `<signature, nonce, simtime>` da `address`

I droni hanno una struttura dati interna, `publicKeyStationMap<int, vector<unsigned char>>` che memorizza le chiavi pubbliche di tutte le stazioni: recupereranno la chiave in base alla loro posizione. Tramite un metodo, `getStationIndex`, i droni recuperano l'indice della stazione del loro quadrante, utilizzando poi quest'informazione per recuperare la chiave relativa. A loro volta le stazioni possiedono una struttura dati interna, `publicKeyDroneMap<int, vector<unsigned char>>` che permette loro di memorizzare le chiavi pubbliche dei droni presenti nel quadrante di loro responsabilità.

## 10.2 Diagramma di sequenza della mitigazione

Di seguito, un sequence diagram di esempio per comprendere come funziona la verifica della firma. Possiamo osservare diversi punti chiave:

1. le stazioni malevole non si preoccupano di verificare nessuna firma, ma ogni qualvolta ricevono una richiesta, che sia una `PositionRequest` o una `S2SPositionRequest`, si comportano allo stesso modo: generano una posizione falsa e tentano di generare una firma falsa;
2. tutte le altre stazioni verificano la firma delle altre stazioni ogni qual volta vengono contattate da queste, è bene notare che in questo caso lo scenario è stato "rilassato": poiché gli unici attori malevoli sono le stazioni, non si è ritenuto necessario verificare, almeno in questo caso, le firme dei droni;
3. i droni verificano prima la firma della stazione, dopo la firma del drone;; le firme dei droni in questo caso sono verificate per una questione di consistenza dei dati;
4. le stazioni malevole cercano di generare una firma falsa da passare come `droneSignature` al drone richiedente: questo è necessario perché avendo generato una posizione falsa, la `droneSignature` precedente sicuramente non sarà più valida;

5. se le stazioni non riescono a verificare una firma, ignorano la risposta;
6. se i droni non riescono a verificare una firma, ritardano il routing del pacchetto.

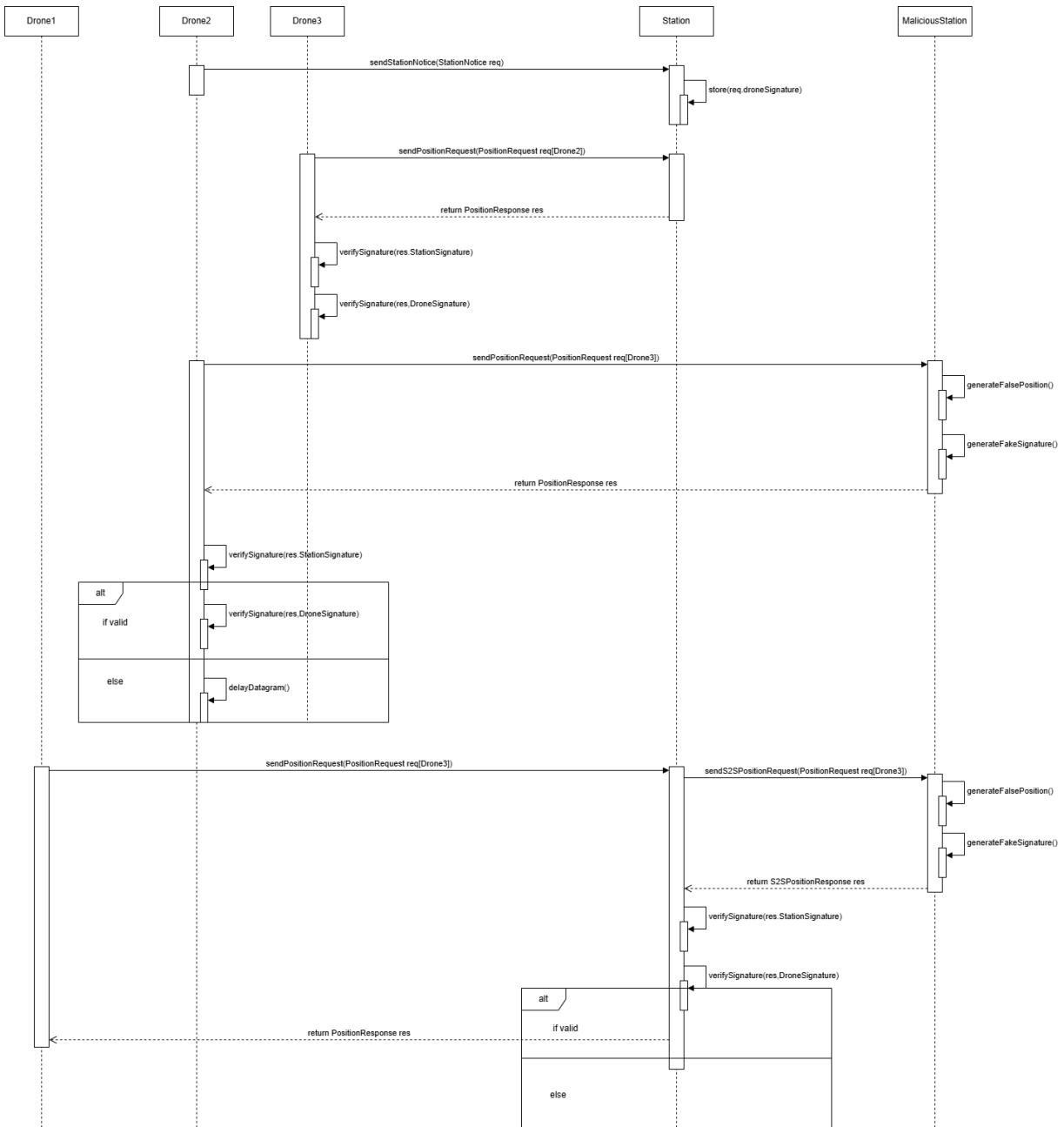


Figure 123: Sequence Diagram

### 10.3 Limitazioni dei danni

Occorre successivamente definire il comportamento dei droni e delle stazioni una volta identificate le stazioni malevole. Le stazioni si limiteranno ad ignorare qualsiasi risposta con una firma non verificata, evitando di inoltrare ai propri droni informazioni errate. Per i droni, la situazione è più complessa, poiché essi possono comunicare esclusivamente con la stazione più vicina. L'idea implementata è stata quella di far sì che ogni volta che un drone non riesce a verificare una firma di una stazione, esso ritardi l'instradamento del pacchetto fino a quando non entrerà nel quadrante di una stazione non malevola, appoggiandosi ad una struttura dati apposita  $multimap<L3Address, Packet>$  targetAddressToDelayedPacket. Si noti che quest'approccio è mirato a mantenere coerenza nel protocollo GPSR e non necessariamente a ridurre la latenza della rete: un drone infatti, potrebbe non uscire mai dal quadrante di una stazione malevola. Dalle simulazioni effettuate risulta che quest'approccio migliori comunque il funzionamento generale della rete.

---

**Algorithm 25** Generazione chiavi

---

```
1: Output: PublicKey  $K_{pb}$ , PrivateKey  $K_{pr}$ 
2: if ECDSA then
3:   generateKeysECDSA()
4: else
5:   generateKeysEDDSA()
6: end if
```

---

---

**Algorithm 26** Firma di un messaggio

---

```
1: Input: String  $message$ , PrivateKey  $K_{pr}$ 
2: Output: String  $signature$ , uint64  $nonce$ 
3:  $nonce \leftarrow generateNonce()$ 
4:  $toSign = hash(message + nonce)$ 
5:  $signature \leftarrow sign(toSign, K_{pr})$ 
6: return ( $signature, nonce$ )
```

---

---

**Algorithm 27** Verifica di una firma

---

```
1: Input: String  $clear$ , L3Address  $source$ , String  $signature$ , uint64  $nonce$ 
2: Output: bool  $verified$ 
3:  $publicKey \leftarrow publicKeyMap[source]$ 
4:  $digest \leftarrow hash(clear + nonce)$ 
5: return  $verify(digest, signature, publicKey)$ 
```

---

---

**Algorithm 28** Processamento di StationNotice - Mitigazione

---

```
1: Input: bool dereg, L3Address source, Coord position, string signature, uint64 nonce
2: if dereg then
3:   quadNodesPositionTable.remove(source)
4: else
5:   if verifySignature(source, signature) then
6:     quadNodesPositionTable.add(source, position)
7:     time = SimTime.now()
8:     quadNodesSignatureMap[source] = (signature, nonce, time)
9:   end if
10: end if
```

---

---

**Algorithm 29** Processamento di PositionRequest - Mitigazione

---

```
1: Input: L3Address destination
2: Output: Coord pos, bool setted, String droneSignature, uint64 droneNonce, String stationSignature, uint64 droneSignature
3: if quadNodesPositionTable.has(destination) OR isMalicious then
4:   if isMalicious then
5:     pos  $\leftarrow$  generateFalsePosition()
6:     clear  $\leftarrow$  pos.x + pos.y + pos.z + " "
7:     droneNonce  $\leftarrow$  generateNonce()
8:     droneSignature  $\leftarrow$  generateFakeSignature(clear, nonce)
9:     stationNonce  $\leftarrow$  generateNonce()
10:    stationSignature  $\leftarrow$  generateFakeSignature(clear, stationNonce)
11:    setted  $\leftarrow$  True
12:  else
13:    pos  $\leftarrow$  quadNodesPositionTable.get(destination)
14:    droneSignature  $\leftarrow$  quadNodesSignatureMap[destination].signature()
15:    droneNonce  $\leftarrow$  quadNodesSignatureMap[destination].nonce()
16:    clear  $\leftarrow$  pos.x + pos.y + pos.z
17:    stationNonce  $\leftarrow$  generateNonce()
18:    stationSignature  $\leftarrow$  signMessage(clear, stationNonce)
19:    setted  $\leftarrow$  True
20:  end if
21: else
22:   setted  $\leftarrow$  False
23: end if
24: return (pos, setted, droneSignature, droneNonce, stationSignature, stationNonce)
```

---

---

**Algorithm 30** Processamento di S2SPositionRequest - Mitigazione

---

```
1: Input: L3Address destination, L3Address applicant
2: Output: Coord pos, bool setted, String droneSignature, uint64 droneNonce, String stationSignature, uint64 droneSignature
3: if quadNodesPositionTable.has(destination) OR isMalicious then
4:   if isMalicious then
5:     pos  $\leftarrow$  generateFalsePosition()
6:     clear  $\leftarrow$  pos.x + pos.y + pos.z
7:     droneNonce  $\leftarrow$  generateNonce()
8:     droneSignature  $\leftarrow$  generateFakeSignature(clear, nonce)
9:     stationNonce  $\leftarrow$  generateNonce()
10:    stationSignature  $\leftarrow$  generateFakeSignature(clear, stationNonce)
11:    setted  $\leftarrow$  True
12:  else
13:    pos  $\leftarrow$  quadNodesPositionTable.get(destination)
14:    droneSignature  $\leftarrow$  quadNodesSignatureMap[destination].signature()
15:    droneNonce  $\leftarrow$  quadNodesSignatureMap[destination].nonce()
16:    clear  $\leftarrow$  pos.x + pos.y + pos.z
17:    stationNonce  $\leftarrow$  generateNonce()
18:    stationSignature  $\leftarrow$  signMessage(clear, stationNonce)
19:    setted  $\leftarrow$  True
20:  end if
21: else
22:   setted  $\leftarrow$  False
23: end if
24: return (pos, setted, droneSignature, droneNonce, stationSignature, stationNonce)
```

---

---

**Algorithm 31** Processamento di PositionResponse - Mitigazione

---

```
1: Input: L3Address destination, Coord pos, String droneSignature, uint64 droneNonce, String stationSignature, uint64 stationNonce
2: if setted then
3:   clear  $\leftarrow$  pos.x + pos.y + pos.z
4:   stationKey  $\leftarrow$  publicKeyMap[getStationIndex(myPos)]
5:   stationVerified  $\leftarrow$  verifySignature(stationKey, clear, signature, stationNonce)
6:   droneKey  $\leftarrow$  publicKeyMap[destination]
7:   droneVerified  $\leftarrow$  verifySignature(droneKey, clear, droneSignature, droneNonce)
8:   if stationVerified AND droneVerified then
9:     routeDatagram()
10:   else
11:     delayDatagram()
12:   end if
13: end if
```

---

---

**Algorithm 32** Processamento di S2SPositionResponse - Mitigazione

---

```
1: Input: L3Address destination, L3Address applicant, Coord pos, String droneSignature, uint64
   droneNonce, String stationSignature, uint64 stationNonce
2: Output: bool isSetted, L3Address destination, Coord pos, simtime time, droneSignature, uint64
   droneNonce, String stationSignature, uint64 stationNonce
3: if setted then
4:   clear  $\leftarrow$  pos.x + pos.y + pos.z
5:   stationKey  $\leftarrow$  publicKeyMap[applicant]
6:   stationVerified  $\leftarrow$  verifySignature(stationKey, clear, signature, stationNonce)
7:   droneKey  $\leftarrow$  publicKeyMap[destination]
8:   droneVerified  $\leftarrow$  verifySignature(droneKey, clear, droneSignature, droneNonce)
9:   if stationVerified AND droneVerified OR isMalicious then
10:    if isMalicious then
11:      pos  $\leftarrow$  generateFalsePosition()
12:      clear  $\leftarrow$  pos.x + pos.y + pos.z
13:      droneNonce  $\leftarrow$  generateNonce()
14:      droneSignature  $\leftarrow$  generateFakeSignature(clear, nonce)
15:      stationNonce  $\leftarrow$  generateNonce()
16:      stationSignature  $\leftarrow$  generateFakeSignature(clear, stationNonce)
17:      setted  $\leftarrow$  True
18:    else
19:      pos  $\leftarrow$  quadNodesPositionTable.get(destination)
20:      droneSignature  $\leftarrow$  quadNodesSignatureMap[destination].signature()
21:      droneNonce  $\leftarrow$  quadNodesSignatureMap[destination].nonce()
22:      clear  $\leftarrow$  pos.x + pos.y + pos.z
23:      stationNonce  $\leftarrow$  generateNonce()
24:      stationSignature  $\leftarrow$  signMessage(clear, stationNonce)
25:      setted  $\leftarrow$  True
26:    end if
27:  end if
28: end if
29: return (pos, setted, droneSignature, droneNonce, stationSignature, stationNonce)
```

---

## 10.4 Valutazione delle performance

I parametri di simulazione adottati sono riportati nella Tabella 12.

Parametro	Valore
Area di simulazione	$1000 \times 1000$ m
Numero totale di nodi	100
Numero minimo di stazioni malevole	0
Numero massimo di stazioni malevole	4
Tempo di simulazione	60/120 s
Algoritmo di firma	ECDSA / EDDSA
Modalità di mobilità	RandomWaypointMobility
Bitrate di trasmissione	24 Mbps
Intervallo dei beacon	1.0 s
Raggio di trasmissione	250m
Capacità nominale della batteria	55.5 Wh
Modalità di invio	FixedMaximum (100 messaggi) / Interval (variabile)
Minimo intervallo di invio	5 s
Massimo intervallo di invio	10 s
Numero di simulazioni	40

Table 12: Parametri della simulazione

#### 10.4.1 60 secondi

Confrontiamo adesso i risultati ottenuti dalle simulazioni degli attacchi con quelli ottenuti dalle simulazioni con la mitigazione. Viene introdotta una nuova misura relativa al consumo dei droni, al fine di valutare l'impatto energetico delle contromisure adottate. Per prima cosa visulizziamo i grafici con l'algoritmo di firma ECDSA.

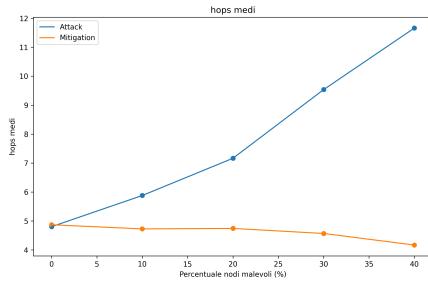


Figure 124: Mean Hops (60 s)

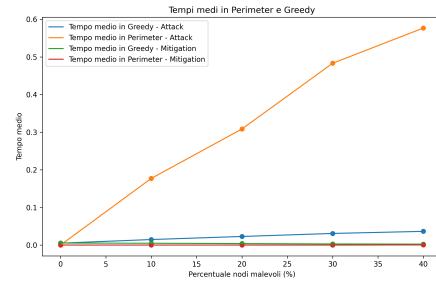


Figure 125: Per Greedy Time (60 s)

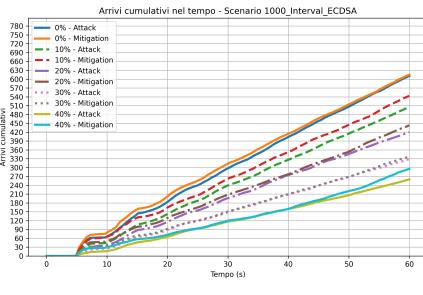


Figure 126: Arrivi cumulativi nel tempo (60 s)

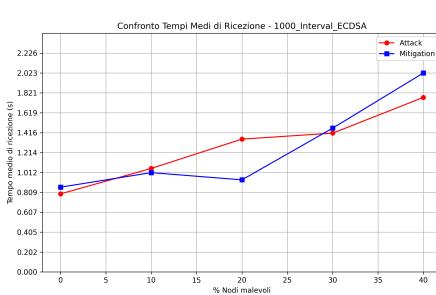


Figure 127: Reception Time (60 s)

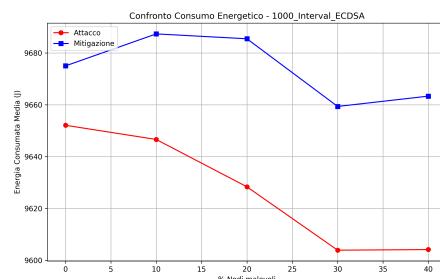


Figure 128: Consumo di energia (60 s)

## Utilizzando l'algoritmo di firma EDDSA

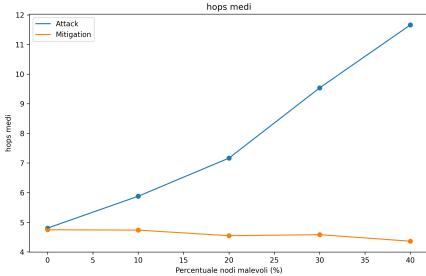


Figure 129: Mean Hops (60 s)

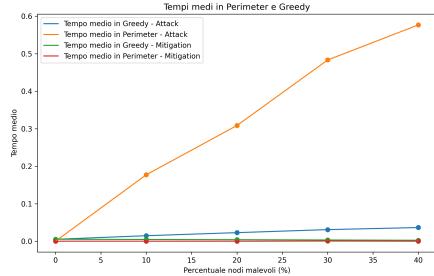


Figure 130: Per Greedy Time (60 s)

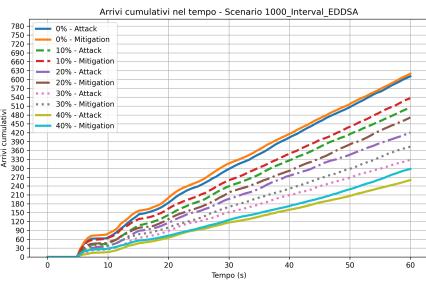


Figure 131: Arrivi cumulativi nel tempo (60 s)

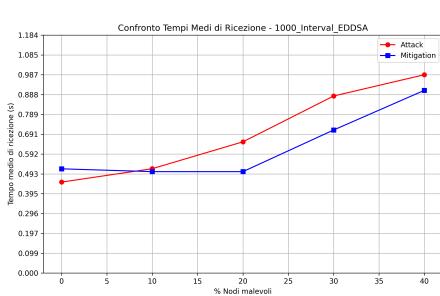


Figure 132: Reception Time (60 s)

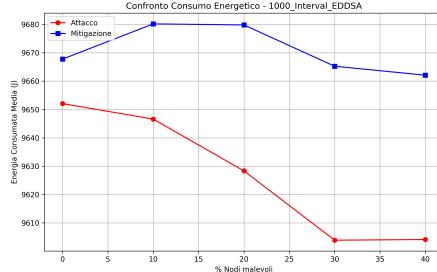


Figure 133: Consumo di energia (60 s)

Si può notare dai grafici come l'adozione delle mitigazioni migliori significativamente le prestazioni del protocollo:

- **Diminuzione del numero degli hop:** la strategia di inviare esclusivamente ai nodi con posizioni consistenti consente di mantenere una piena coerenza nel protocollo GPSR.
- **Maggiore numero di pacchetti ricevuti:** ignorando l'operato delle stazioni malevole, è

possibile garantire una piena coerenza nel resto dellos scenario, permettendo a tutti i pacchetti iviati al di fuori del quadrante di una stazione di arrivare abilmente a destinazione.

- **Diminuzione del tempo in modalità *perimeter*:** la piena coerenza del GPSR riduce la necessità di attivare il *perimeter forwarding*.

Questi benefici si osservano in entrambe le modalità di firma digitale, sia con **ECDSA** che con **EDDSA**, confermando l'efficacia del meccanismo di mitigazione indipendentemente dall'algoritmo crittografico adottato. In contrapposizione, si evince un aumento dei consumi dovuto alle attività di cifratura e decifratura dei nodi.

#### 10.4.2 120 secondi

L'estensione del tempo di simulazione a 120 secondi conferma i benefici osservati con le mitigazioni. Anche in questo scenario, si registra un aumento del numero di pacchetti consegnati, una riduzione significativa del tempo medio di ricezione e del tempo trascorso in modalità *perimeter*. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano la stabilità e l'efficacia del protocollo di mitigazione su periodi di simulazione più lunghi.

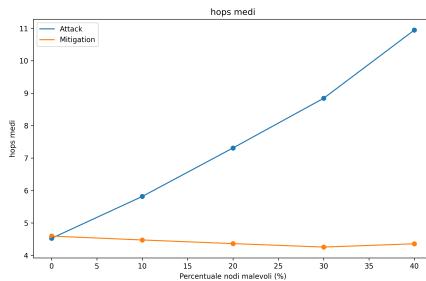


Figure 134: Mean Hops (120 s)

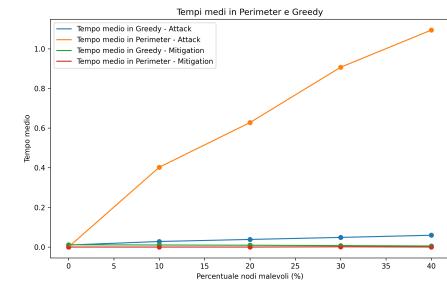


Figure 135: Per Greedy Time (120 s)

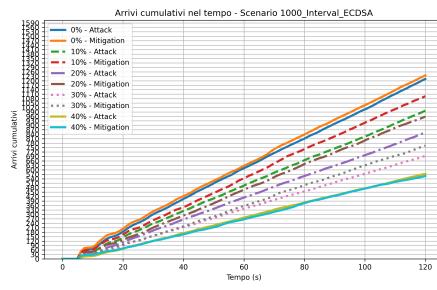


Figure 136: Arrivi cumulativi nel tempo (120 s)

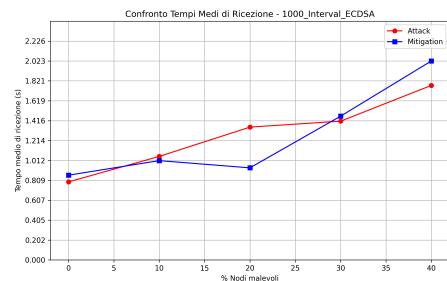


Figure 137: Reception Time (120 s)

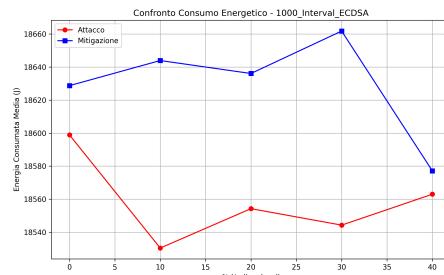


Figure 138: Consumo di energia (120 s)

Utilizzando l'algoritmo di firma EDDSA

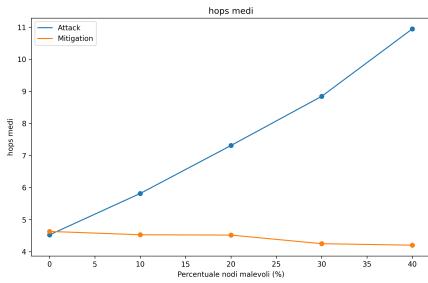


Figure 139: Mean Hops (120 s)

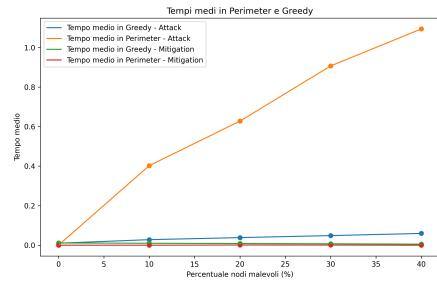


Figure 140: Per Greedy Time (120 s)

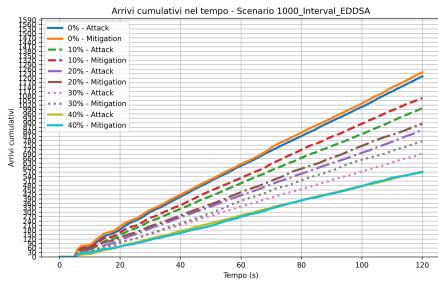


Figure 141: Arrivi cumulativi nel tempo (60 s)

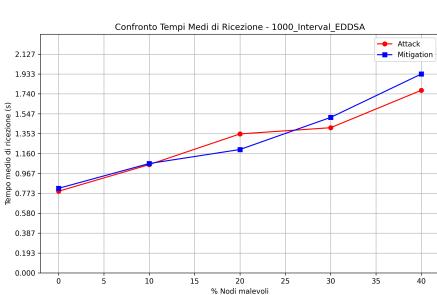


Figure 142: Reception Time (120 s)

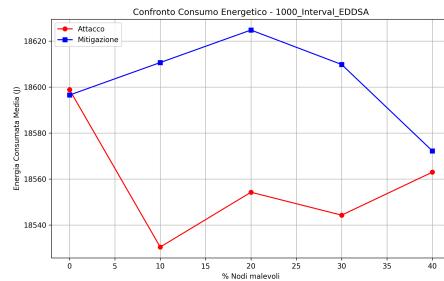


Figure 143: Consumo di energia (120 s)

Tuttavia, il consumo energetico risulta maggiore rispetto allo scenario senza mitigazioni, poiché i nodi eseguono operazioni aggiuntive per il controllo e l'esclusione dei nodi malevoli. Questi risultati, validi per entrambe le modalità di firma **ECDSA** ed **EDDSA**, dimostrano l'efficacia del protocollo di mitigazione, pur con un costo energetico aggiuntivo.

#### 10.4.3 Confronto fra ECDSA e EDDSA 60 secondi

Vediamo ora come differiscono i due algoritmi di firma digitale proposti, **ECDSA** ed **EDDSA**, valutando l'impatto che questi hanno sulle simulazioni avviate con i parametri precedenti, seguendo le stesse modalità di avvio, inserendo anche il *tempo medio di firma e verifica* per evidenziare meglio l'efficienza computazionale dei due algoritmi.

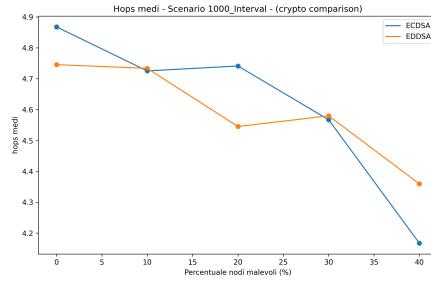


Figure 144: Mean Hops (60 s)

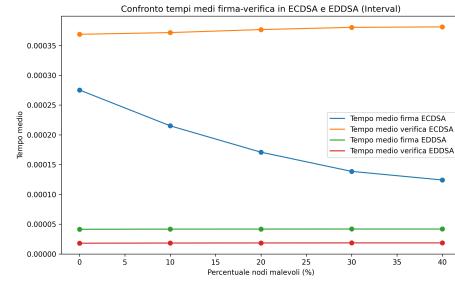


Figure 145: Tempo di firma-verifica (60 s)

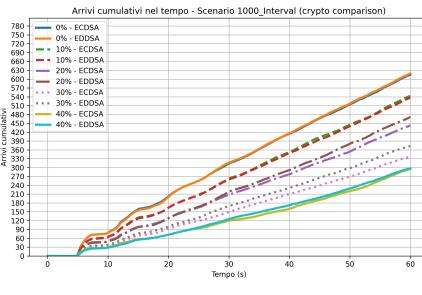


Figure 146: Arrivi cumulativi nel tempo (60 s)

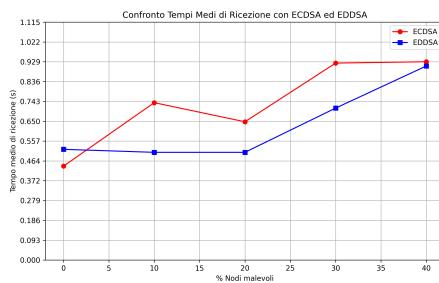


Figure 147: Reception Time (60 s)

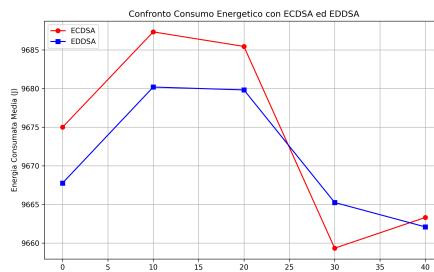


Figure 148: Consumo di energia (60 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** EDDSA si dimostra più veloce rispetto a ECDSA,

riducendo il tempo necessario per la generazione e la verifica delle firme digitali.

- **Numero medio di hop:** i due algoritmi si dimostrano entrambi abbastanza simili dal punto di vista di questa metrica, con EDDSA leggermente meglio all'aumentare del numero di malevoli, ECDSA meglio al diminuire.
- **Consumo energetico:** EDDSA risulta più efficiente dal punto di vista energetico.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono un numero di pacchetti consegnati simile, senza differenze significative.
- **Tempo medio di ricezione:** le due modalità di firma presentano valori comparabili, con dei valori leggermente a favore di EDDSA.

#### 10.4.4 Confronto fra ECDSA e EDDSA 120 secondi

Estendendo il tempo di simulazione a 120 secondi, si osserva che le differenze tra **ECDSA** ed **EDDSA** tendono a confermarsi su valori simili a quelli osservati in precedenza.

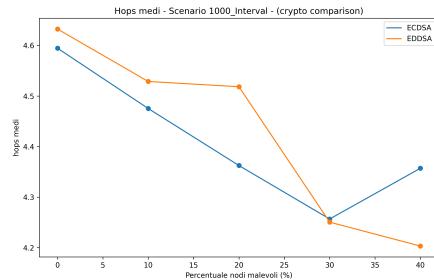


Figure 149: Mean Hops (120 s)

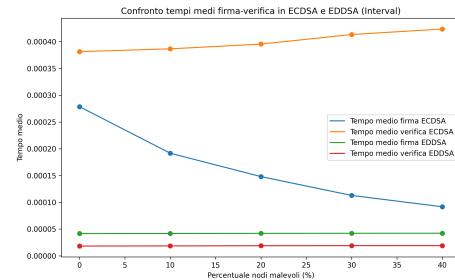


Figure 150: Tempo di firma-verifica (120 s)

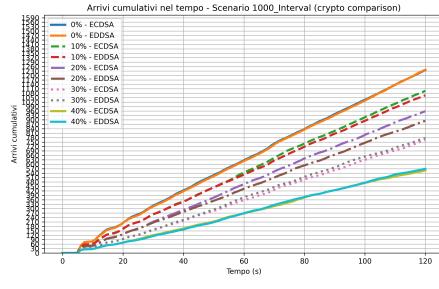


Figure 151: Arrivi cumulativi nel tempo (120 s)

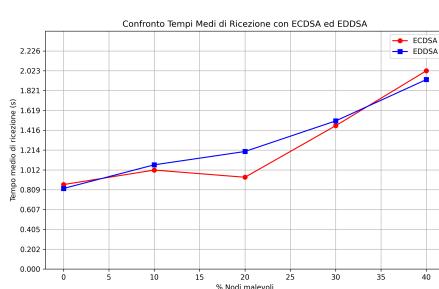


Figure 152: Reception Time (120 s)

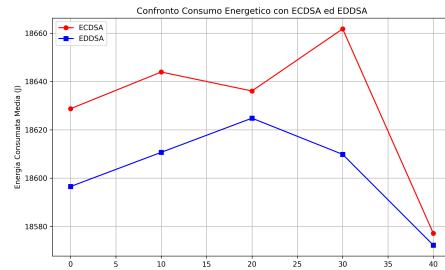


Figure 153: Consumo di energia (120 s)

Dai risultati ottenuti emerge che:

- **Tempo medio di firma e verifica:** si riconferma l'osservazione fatta in precedenza con i due algoritmi che presentano tempi di firma e verifica analoghi.

- **Consumo energetico:** il dispendio energetico tra ECDSA ed EDDSA risulta confermarsi sui valori osservati nelle simulazioni precedenti.
- **Numero cumulativo di pacchetti ricevuti:** entrambi gli algoritmi garantiscono una quantità di pacchetti ricevuti molto simile, senza evidenti differenze in termini di affidabilità della trasmissione.
- **Tempo medio di ricezione:** i tempi medi di consegna dei pacchetti rimangono simili tra ECDSA ed EDDSA.
- **Numero medio di hop:** anche il numero di hop risulta molto simile a quello che abbiamo osservato nel batch di simulazione precedente.

Questi risultati suggeriscono che, con tempi di simulazione più lunghi, le performance dei due algoritmi si mantengono stabili, confermando nella quasi totalità dei casi ciò che abbiamo osservato con le simulazioni più brevi.

## 10.5 Conclusioni

Il sistema di mitigazione proposto è molto efficiente nell'identificare le stazioni malevole e che non forniscono dati consistenti, grazie al meccanismo a chiave asimmetrica bilaterale che permette di resistere in modo convincente ai tentativi di tampering effettuati dalle stazioni. Tuttavia, è bene notare che i droni hanno comunque una finestra di azione limitata: l'unico modo per isolare una stazione malevola è uscire dal suo quadrante. Abbiamo notato dalle simulazioni effettuate che quest'approccio sul lungo periodo non compromette più di tanto la latenza della rete, permettendo di avere prestazioni superiori rispetto allo scenario con solo l'attacco. Inoltre, permette di garantire una piena coerenza del protocollo GPSR, riducendo vertiginosamente il numero di hops di un pacchetto.

## References

- [1] B. Karp and H. T. Kung, “Gpsr: Greedy perimeter stateless routing for wireless networks,” in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*. ACM, 2000, pp. 243–254.
- [2] G. Danezis, P. Mittal, and A. Bursztein, “The sybil attack in sensor networks: Analysis and mitigation,” in *Proceedings of the 1st IEEE International Conference on Pervasive Computing and Communications*. IEEE, 2004, pp. 31–38.
- [3] P. Rani, Kavita, S. Verma, and G. N. Nguyen, “Mitigation of black and gray hole attacks using swarm inspired algorithm,” *IEEE Access*, vol. 8, pp. 121 755–121 764, 2020.
- [4] M. Tropea, M. G. Spina, A. Lakas, P. Sarigiannidis, and F. D. Rango, “Secgpsr: A secure gpsr protocol for fanet against sybil and gray hole attacks,” *IEEE Access*, vol. 12, pp. 186 909–186 923, 2024.
- [5] C. Lin, D. He, N. Kumar, K. R. Choo, A. Vinel, and X. Huang, “Security and privacy for the internet of drones: Challenges and solutions,” *IEEE Communications Magazine*, vol. 56, no. 1, pp. 64–69, 2018.
- [6] F. Pasandideh, J. P. J. da Costa, R. Kunst, N. Islam, W. Hardjawana, and E. P. de Freitas, “A review of flying ad hoc networks: Key characteristics, applications, and wireless technologies,” *Remote Sensing*, vol. 14, no. 18, p. 4459, 2022.