

COMMUNICATION PROTOCOL FOR UTILITY NETWORKS

Authors:

Michael Vasquez Otazu

Massimo Grilli

2017

Table of Contents

1. INTRODUZIONE.....	4
2. DESCRIZIONE.....	5
2.1 SERVIZI.....	5
2.1.1 Hello	5
2.1.2 Update Noise.....	6
2.1.3 Request Noise Table	7
2.1.4 Send Response	7
2.1.5 Watchdog	8
2.2 PACCHETTO.....	8
2.2.1 Preambolo	9
2.2.2 PDU	9
2.2.3 Header	10
2.2.4 Payload	10
2.3 ECC.....	13
2.2 FUNZIONAMENTO	14
2.2.1 Codifica e decodifica	14
2.2.2 Consistenza del protocollo.....	14
2.2.3 Priorità	16
2.2.4 Algoritmo d'instradamento	17
2.2.5 Timeout	17
2.3 TABELLE.....	18
2.3.1 Noise Table (Slave).....	18
2.3.2 Noise Table (Master).....	18
3. REQUISITI	19
3.1 Requisiti Funzionali	19
3.2 Requisiti Non-Funzionali	20
4. ANALISI.....	21
4.1 GELLISH	21
4.2 USE CASE	25
4.3 ACTIVITY DIAGRAM	26
4.3.1 AD Hello	26
4.3.2 AD Watchdog	26
4.3.3 SD Update Noise	27

4.3.4	SD Request Noise Table	27
4.3.5	SD Send/Response	28
4.3.6	SD Receive Message (Master).....	28
4.3.7	SD Receive Message (Slave).....	29
4.3.8	SD Dispatch Message (Master).....	29
4.3.9	SD Dispatch Message (Slave)	30
5.	PROGETTAZIONE.....	31
5.1	CLASS DIAGRAM	31
5.2	SEQUENCE DIAGRAM	32
5.2.1	SD Hello	32
5.2.2	SD Update Noise	33
5.2.3	SD Request Noise Table	34
5.2.4	SD Send/Response	36
5.2.5	SD WatchDog.....	38
5.3	OBJECT DIAGRAM.....	39
5.3.1	OD Hello	39
5.3.2	OD Request Noise Table	39
5.3.3	OD Send/Response	40
6.	CONCLUSIONI.....	Error! Bookmark not defined.

1. INTRODUZIONE

L'architettura del progetto si basa sul modello del protocollo di comunicazione “**Master-Slave**”, dove il **Master** (Collettore) richiede o invia dati agli **Slave** (Dispositivi), la comunicazione nell'architettura di riferimento è di tipo **monodirezionale** e preferibilmente del tipo “**bus channel**” (es. RS 485 o Powerline) per la sua scalabilità e convenienza.

Il **meccanismo di connessione** è basato su una rete di distribuzione collegata tramite dei nodi fissi (hop) forniti da un punto d'accesso alla rete e quindi raggiungibili da un collettore (master).

Nello scenario attuale il protocollo comprende l'utilizzo di una singola istanza del **collettore**, che avrà il maggior carico computazionale. In altri scenari, è fattibile la creazione di una federazione di collettori in rete eterogenee (o fuori misura) come un **servizio** a livello OSI 7.

Il **componente elettronico** dei **dispositivi** che implementano fisicamente i nodi consistono principalmente in dei semplici microcontrollori con sensori:

- **Slave**: Arduino
- **Master**: Raspberry

Inoltre, tali dispositivi dovranno essere collegati ad **adattatori Powerline** (a loro volta collegati alle prese) onde poter accedere alla rete elettrica e all'apposito **protocollo Powerline** che implementa i livelli 1° e 2° della ISO/OSI.

Il **linguaggio di programmazione** utilizzato per lo sviluppo dell'architettura software è una versione apposita del linguaggio C, portabile per i dispositivi basati su Arduino (tramite delle opportune librerie e framework).

2. DESCRIZIONE

2.1 SERVIZI

In questa sezione vengono definiti i vari **servizi** del protocollo, basati sullo **scambio di messaggi** (*codificati tramite l'algoritmo di Hamming e opportunamente veicolati tramite l'algoritmo di Dijkstra*) tra il dispositivo **Master** ed i vari dispositivi **Slave**.

2.1.1 Hello

Onde poter utilizzare le specifiche funzioni (*e.g. controllo dei sensori, ecc.*) dei dispositivi **Slave**, il **Master** deve essere capace di tracciarli e veicolarli non appena essi si collegano alla rete e vengono riconosciuti tramite la seguente procedura:

- i. Quando un **nuovo dispositivo** si collega alla rete per la prima volta oppure effettua un riavvio (*in seguito ad una prolungata assenza di comunicazione*), esso invia un **messaggio** (*contenente il suo MAC Address, opportunamente codificato con Hamming*) tramite il **servizio “Hello”** del protocollo.
NB: Il messaggio è creato ed inviato automaticamente dal dispositivo non appena esso viene collegato alla rete elettrica.
- ii. Dopo l'invio del messaggio il **nuovo dispositivo** rimarrà in **attesa** di essere contattato dal **Master**, eseguendo continuamente il **servizio** di protocollo **“Hello”** con periodi asincroni (*onde non rischiare collisioni a loop continuo nel caso ci fossero trasmissioni contemporanee di pacchetti sulla stessa parte di rete*).
- iii. Il messaggio inviato dal **nuovo dispositivo** può giungere al **Master** sia direttamente che tramite i nodi ripetitori.

Finché il messaggio del **nuovo dispositivo** non giunge al **Master**, esso viene inoltrato dai **nodi** (dispositivi già identificati) che si trovano nel *raggio di trasmissione* tramite il servizio **“Hello Forwarding”**, che appende al messaggio originale il MAC Address del **nodo** attuale in modo di permettere al **Master** di ricostruire successivamente il **percorso di routing**.

È opportuno ribadire che l'inoltro del messaggio viene effettuato esclusivamente dai **dispositivi già identificati**, quelli ancora in attesa di essere riconosciti si limiteranno ad ascoltare il messaggio passivamente.

- iv. Una volta che il messaggio sarà giunto al **Master**, nel caso la sua distanza di Hamming rientri nei parametri accettabili, esso aggiungerà il MAC Address del **nuovo dispositivo** nella **“Tabella di Routing”** (rappresentazione topografica della rete), richiedendo successivamente al dispositivo di inviare la sua **“Noise Table” locale** tramite il servizio **“Request Noise Table”** del protocollo.

2.1.2 Update Noise

Il **valore** del “rumore” tra nodi è memorizzato in una **tabella** detta **“Noise table”** come **tuple di rumore** del tipo **<sorgente, destinatario, rumore>**, **aggiornata ad ogni ricezione** di pacchetto attraverso il protocollo di **“Update Noise”** nella seguente maniera:

- i. Ogni **nodo** effettua un continuo rilevamento dei valori dei rumori nei **pacchetti** in transito nella sua **zona di visibilità** (sia nel ruolo di hop, destinatario finale oppure semplice ascoltatore) **decodificandoli e rilevando** poi gli errori eventualmente presenti.
- ii. Ogni **nodo**, dopo aver effettuato il rilevamento, aggiorna di conseguenza la sua **“Noise Table” locale** per poi inviarla al **Master** nei modi seguenti:
 - tramite il servizio di protocollo **“Request Noise Table”**, oppure
 - tramite il servizio di protocollo **“Send Response”**, quando il nodo verrà coinvolto (come hop o destinatario) nel forwarding dei pacchetti.

Questo rilevamento ed aggiornamento continuo permette al **Master** di costruire successivamente e continuamente un percorso affidabile e aggiornato, di pacchetti, tra sé ed i dispositivi attivi sulla rete.

- iii. Oltre ad inserire nella **“Noise Table”** le **tuple** con il **valore del rumore**, viene anche impostato il relativo **“Check bit”** (0: non modificato, 1: modificato) a **1** per evidenziare l’aggiornamento effettuato nella tabella (ed abilitarlo per un futuro invio al **Master**).
- iv. Durante l’esecuzione del servizio di **“Update Noise”** i nodi ascoltano messaggi di ogni tipo e la qualità della ricezione può essere valutata in base al rumore del canale su cui i pacchetti viaggiano tramite **l’algoritmo di Hamming (11,4)** aggiungendo un **“bit di parità”** alla fine (per fornire una maggior sicurezza della quantità di **“dirty bit”**).

2.1.3 Request Noise Table

Dopo che il **Master** assegna un **indirizzo di sottorete** ad un nuovo nodo, esso gli conferma la sua avvenuta registrazione tramite il servizio di “**Request Noise Table**”, questo nel modo seguente:

- i. Ogni volta che il **Master** riceve un messaggio di “**Hello**” proveniente da un **nuovo dispositivo (slave)**, esso comunica al medesimo il suo ingresso nella rete attraverso il servizio di protocollo “**Request Noise Table**”, che consiste nell’invio di un messaggio contenente la richiesta della sua “**Noise Table**” locale.

NB: La suddetta tabella servirà al **Master** per stabilire un primo **percorso di routing** (attraverso i nodi adiacenti) basato sul rumore percepito dal **nuovo dispositivo (slave)**.

- ii. Lo **Slave** risponderà alla richiesta con un messaggio di “**Request Noise Table**” contenente nel **payload** tutte le **tuple di rumore aggiornate** rispetto all’ultimo invio al **Master** (i.e. il cui “**Check bit**” è impostato a 1).

NB: Dopo l’invio, il “**Check bit**” viene ripostato a 0 per prevenire un l’invio ripetitivo di dati.

- iii. Nel caso in cui il **Master** si accorge che il percorso verso un certo **nodo slave** non è aggiornato da un certo tempo (es. 1 ora), esso esegue un’altra volta il servizio di “**Request Noise Table**” ai nodi **adiacenti** al suddetto nodo slave onde verificare la sua disponibilità.

In caso un nodo slave risulti **assente** per più di un determinato lasso di tempo (es. 48 ore), esso verrà rimosso dalla **tavella di routing** insieme ad ogni suo riferimento.

2.1.4 Send Response

Il **controllo dei dispositivi** (rilevatori, attuatori, ecc.) è eseguito dal **Master** tramite l’invio di un messaggio al relativo **nodo (Slave)** utilizzando il protocollo “**Send Response**”, che consiste nelle seguenti operazioni:

- i. L’invio dei **messaggi** da parte del **Master** verso gli **Slave** avviene periodicamente, sia in **background** (e.g. controlli di sicurezza degli apparati, ecc.) che come **risposta ad un comando** utente (e.g. rileva temperature, ecc.).

- ii. Come risposta al suddetto messaggio gli **Slave** rispondono inviando al **Master** un messaggio di **“Send Response”** contenente l'informazione richiesta e opportunamente codificato in base all'algoritmo di Hamming.
- iii. Nel caso che il **Master** verifichi che tabella di rumore sia cambiata (i.e. almeno una delle stringhe di rumore sia stata modificata), allora aggiornerà la “Tabella di Rumore” come da protocollo e di conseguenza il percorso d'istradamento da utilizzare per future comunicazioni.

2.1.5 Watchdog

Il servizio **“Watchdog”** permette ai **nodi** di verificare e/o rinnovare la sua presenza nella rete nel modo seguente:

- i. Ogni **Slave** verifica periodicamente il “timestamp” dell'ultimo messaggio in cui è stato coinvolto dal **Master**, calcolando il tempo trascorso da allora senza aver ricevuto altri messaggi.
- ii. Se lo **Slave** riscontra che questo tempo superi il periodo massimo di tolleranza (**timeout**), darà per certo il fatto di essere stato eliminato dalla “Noise Table” del **Master** (e.g. dovuto a problematiche nella comunicazione).
- iii. A questo punto, il dispositivo farà un “auto-reset” e procederà al rinvio di un messaggio di **“Hello”** verso tutti i dispositivi in ascolto (per cui si ha un rumore sotto il livello di guardia).

NB: Il **watchdog** riguarda solo i nodi. Nel caso del **master**, ogni volta esso riscontrerà che un nodo non è più raggiungibile lo cancellerà dalla sua **Noise Table**.

2.2 PACCHETTO

I **pacchetti** che navigano nella rete sono i componenti essenziali per lo **scambio di messaggi** tra i dispositivi e la conseguente costruzione del protocollo. La struttura dei **pacchetti** è definita in modo generico tramite i seguenti **frame**:



La **catena seriale** di pacchetti “**tipo**” di cui sono composti i messaggi, possono possedere **diversi formati** a seconda del servizio di protocollo.

Il **treno di impulsi** di cui i “pacchetti” sono la vista a livello ISO più alto, sarà perciò modulato in gruppi di 16 bit (word) trasferiti progressivamente e in maniera seriale lungo il canale trasmisivo.

2.2.1 Preambolo

La parte iniziale di un pacchetto è il cosiddetto “**preambolo**”, strutturato nella seguente maniera:

PREAMBOLO		
Cicli di ascolto	Stringa di prenotazione	Cicli di ascolto
----- 4 bit -----	----- 8 bit -----	----- 4 bit -----

Lo scopo dell'utilizzo del **preambolo** consiste nella **verifica della disponibilità** della rete per l'invio dei pacchetti, che avviene tramite le seguenti fasi:

i. **Ascolto**

Ricezione di 4 bit (1 bit per ogni **ciclo di clock**), che servono a verificare la presenza o meno di traffico di pacchetti all'interno della rete.

ii. **Prenotazione**

Invio di una stringa di 8 bit alternati (10101010) con lo scopo di comunicare ad altri dispositivi la volontà di occupare la rete.

iii. **Ascolto**

Ricezione di 4 bit (1 bit per ogni **ciclo di clock**), che servono a verificare la presenza o meno di traffico di pacchetti all'interno della rete, dopo che è stata richiesta la prenotazione. Se la rete è libera si procede con la trasmissione del messaggio.

2.2.2 PDU

Il **PDU** (Packet Data Unit) è l'unità d'informazione scambiata, nel protocollo di comunicazione esso è composto dall'**Header** e dal **Payload** che sono opportunamente descritti a continuazione.

2.2.3 Header

L'**Header** gestisce gli stessi campi per tutti i servizi e comandi del sistema; esso è strutturato nel modo seguente:



Data Field	Data Size (bits)	Data Value
PDU Type	1	0: servizio di protocollo 1: comando
Operation ID	$\sum_{i=0}^c f_i$ <i>C: numero di categorie di operazioni</i> <i>f_i: numero max. di funzioni della categoria "i"</i>	Numero del servizio: 1: Hello 01: Req. Noise Oppure di comando. E.g.: 0: Comando X ...

2.2.4 Payload

Il “**Payload**” è la sezione contenente i dati che interessano direttamente i **servizi** del protocollo e **comandi**. Esso è implementato specificamente per ogni **servizio (o comando)** come riportato a continuazione:

i. Hello (Terminal-to-Master)

Data Field	Data Size (bits)	Data Value
MAC Address	48	MAC Address del dispositivo richiedente
TimeToLive	h_{max} : numero max. di hop (1 – 255)	Numero massimo di hop che possono inoltrare il messaggio.

ii. **HelloForwarding** (Hop-to-Master)

Data Field	Data Size (bits)	Data Value
MAC Address	$48 * h$ h : numero di hop da attraversare (1 - 255)	Uno o più MAC Address (di 48 bit) concatenati in ordine di attraversamento
TimeToLive	h_{max} : numero max. di hop (1 – 255)	Numero massimo di hop che possono inoltrare il messaggio (decrementato di 1 ad ogni hop)

iii. **RequestNoiseTable** (Master-to-Terminal)

Data Field	Data Size (bits)	Data Value
Route Offset	h_{max} h_{max} : numero max. di hop (1 – 255)	Numero del prossimo hop da attraversare. NB: Questo valore è continuamente aggiornato dal ultimo hop attraversato.
Routing Path	$48 * h$ h : numero di hop da attraversare (1 - 255)	Uno o più MAC Address (di 48 bit) concatenati in ordine di attraversamento

iv. Request Noise Table (Terminal-to-Master)

Data Field	Data Size (bits)	Data Value
Routing Path	$48 * h$ h : numero di hop da attraversare (1 - 255)	Uno o più MAC Address (di 48 bit) concatenati in ordine di attraversamento. NB: Percorso utilizzato dal Master, ma invertito.
Noise Table	$h * (48 + 1 + 7)$ h : numero di hop nella tabella 48 : MAC Address del hop 1 : bit di controllo di modifica 7 : valore del rumore	Tabella di rumori

v. Send/Response (Master-to-Terminal)

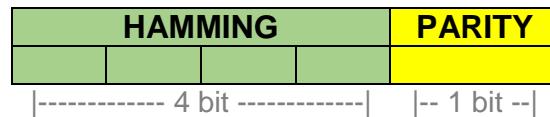
Data Field	Data Size (bits)	Data Value
Route Offset	h_{max} h_{max} : numero max. di hop (1 – 255)	Numero del prossimo hop da attraversare
Routing Path	$48 * h$ h : numero di hop da attraversare (1 - 255)	Uno o più MAC Address (di 48 bit) concatenati in ordine di attraversamento
Command Parameter	p p : numero di bit definito dal comando	Eventuali parametri per la richiesta del comando

vi. **Send Response** (Terminal-to-Master)

Data Field	Data Size (bits)	Data Value
Route Offset	h_{max}	Numero del prossimo hop da attraversare h_{max} : numero max. di hop (1 – 255)
Routing Path	$48 * h$	Uno o più MAC Address (di 48 bit) concatenati in ordine di attraversamento h : numero di hop da attraversare (1 - 255) NB: Percorso utilizzato dal master, ma invertito.
Requested Information	p	Eventuali informazioni richiesta dal comando p : numero di bit definito dal comando
Noise Tables	$(r * (48 + 1 + 7)) * h$	Tabelle di rumore caricate da ogni hop nel percorso (solo valori cambiati) r : numero di righe nella tabella (ogni hop) 48 : MAC Address del hop 1 : bit di controllo di modifica 7 : valore del rumore h : numero di hop da attraversare (1 - 255)

2.3 ECC

L'Error Correction Code serve a rilevare e correggere errori nei pacchetti:



2.2 FUNZIONAMENTO

2.2.1 Codifica e decodifica

L'informazione trasmessa nel sistema sotto forma di pacchetti è opportunamente **codificata** (*specificamente a seconda del servizio o comando di riferimento*) per garantire la sua integrità.

Una volta che il pacchetto è ricevuto dal master oppure un nodo, esso viene **decodificato** per estrapolare i **dati** rilevanti (e.g. Servizio di riferimento, MAC Address, Routing Path, Parametri, ecc.), tenendo conto del controllo di un eventuale **deterioramento** dell'informazione (*attraverso l'algoritmo di decodifica di Hamming*).

2.2.2 Consistenza del protocollo

2.2.2.1 Rilevamento e Correzione degli Errori

Tutti i dispositivi (Master e Slave) codificano e decodificano i messaggi tenendo conto della rilevazione ed eventuale correzione di bit errati nei pacchetti che transitano sul tratto di rete su cui tali dispositivi sono in ascolto, questo tramite l'algoritmo della **Distanza di Hamming**. Tutti i **pacchetti** non saranno altro che **gruppi di word** (16 bit) opportunamente **codificati** prima della trasmissione tramite **l'Algoritmo della distanza di Hamming (11,15) + 1** e **decodificati** successivamente.

Per ogni pacchetto è tollerato un certo **margine di errore** sui bit, nella misura che l'informazione possa essere completamente ricostruita. Questo garantisce una buona individuazione e a volte la correzione stessa sotto un certo numero di bit invalidati, cioè di eventuali alterazioni sui **bit pattern** di cui sono composti i pacchetti. Ogni nodo che stia eseguendo il servizio di **“Send Response”** dovrà sempre effettuare la **codifica di Hamming** sui “bit utili” del messaggio che gli è arrivato, aggiornando successivamente le **differenze** nella “Tabella di Rumori” attraverso l'opportuno servizio (Update Noise).

Qualora l'errore rilevato fosse solo uno il bit dove vi è l'errore viene corretto ed il messaggio viene inoltrato all'hop successivo nel percorso di routing. Invece, se gli errori sono più di uno, verrà inviato un messaggio di “decoding error” specificando l'incapacità di ricavare dati consistenti dal messaggio ricevuto.

2.2.2.2 Messaggio di Time-out Error

Nel caso in cui il **tempo di attesa** (*calcolato in base alle specifiche del capitolo 4.6*) di un **messaggio** instradato in rete superasse il **limite massimo** (*stabilito a seconda del proprio percorso di routing*), il **hop** che rileva per primo tale **ritardo** comunicherà (al **hop precedente** nel percorso di routing) l'impossibilità di inviare il messaggio nei tempi stabiliti dal protocollo. Il **hop precedente**, a sua volta, in modalità “a cascata” dovrà fare il **forwarding** del messaggio al **hop** che lo precede, fino ad arrivare al master.

Questo **messaggio**, denominato “**Time-out Error**”, è un meccanismo di avvertimento permette di far fronte al **superamento del tempo massimo di attesa**, e fa parte del protocollo implementato in **tutti gli hops** in modo che il trasferimento dei pacchetti sia eseguito e sincronizzato sempre correttamente.

Il **messaggio di “Time-out Error”** è più corto e quindi più veloce da creare e inviare in rete agli **hop** del percorso di routing, poiché conterrà solamente i dati seguenti:

Codice Errore	MAC “n”	MAC “n-1”	...	MAC 1
---------------	---------	-----------	-----	-------

- **Codice errore**: relativo al messaggio in questione.
- **MAC “n”**: Identificativo MAC del **hop rilevatore (mittente)**.
- **MAC “n-1” ... 1**: Estrapolazione (eseguita da ogni hop) del **percorso di routing**, contenente (a ritroso) gli identificativi MAC degli **hop** che ne fanno parte, da quello corrente fino a quello finale.

Inoltre, il **messaggio di “Time-out Error”** serve al **Master** ad evitare l'**hop malfunzionante** (i.e. per cui è stato rilevato un ritardo) e di individuare facilmente **percorsi alternativi** (cammini minimi più efficienti) attraverso l'opportuno **Algoritmo di instradamento** sempre in base alla sua **Noise Table**.

Nel caso non esistano percorsi alternativi, il **Master** eseguirà le dovute azioni in base alle specifiche del protocollo (e.g. comunicare alla rete la presenza di un “nodo irraggiungibile” dovuto a errori nel percorso).

2.2.2.3 Messaggio di Feedback

Qualora non si verificasse nessuna problematica (i.e. decodifica errata o timeout), il messaggio giungerà al nodo destinatario e la comunicazione si concluderà con **successo**.

Dopo la conclusione, il **nodo destinatario** invierà un “**messaggio di feedback**” contenente nel suo “payload” tutti i **valori di ritorno** richiesti dal **messaggio** ricevuto dal Master, nonché le **tuple** della **Noise Table (locale)** che sono state modificate recentemente, azione quest’ultima che verrà eseguita anche da tutti gli altri nodi appartenenti (a ritroso) al percorso di routing.

2.2.3 Priorità

Il protocollo considera delle **politiche di priorità** che permettono di gestire prima i servizi del protocollo stesso davanti ad eventuali comandi ai dispositivi, questo, basato sul campo “**PDU Type**” del pacchetto che può avere i seguenti valori:

Priorità	PDU Type	Scopo
Alta	0	Servizio di protocollo “Send response” (Altissima)
		Servizio di protocollo “Request Noise Table” (Molto alta)
		Servizio di protocollo “Update Noise” (Alta)
Media	1	Commando al dispositivo, e.g.: <ul style="list-style-type: none"> • Accensione luci • Spegnimento condizionatore • Rilevazione temperatura • Ecc. NB: Tutti i comandi hanno la stessa priorità, e sono eseguiti in ordine di arrivo.
Bassa	0	Servizio di protocollo di “Hello Forwarding”

Le suddette politiche di priorità permettono di mantenere la **consistenza** nel funzionamento del protocollo stesso del sistema.

2.2.4 Algoritmo d'istradamento

I percorsi ottimali tra il **collettore** (master) ed i **dispositivi** (slave) sono determinati dal master tramite l'**algoritmo di Dijkstra**, utilizzando come discriminante principale il **rumore** tra i nodi.

I dati necessari per il suddetto calcolo vengono prelevati dalla cosiddetta **“Tabella di Rumore”** gestita dal **Master**, che permette di avere una mappa molto dettagliata dei nodi presenti nella rete e dei rumori da essi percepiti.

2.2.5 Timeout

Onde **minimizzare** il tasso di fallimento nell'invio dei messaggi tramite i servizi del protocollo “Send Response” e “RequestNoiseTable” e favorire il riutilizzo dell'informazione sui fallimenti per migliorare il **calcolo dei percorsi d'istradamento**.

Prima d'inviare un messaggio, il Master e ogni Slave specificato nel percorso fa una estima di **un time-out** basato sulla seguente formula:

$$\text{Timeout} = N * (V + T)$$

- **N:** Il **numero di HOP** necessari per raggiungere il dispositivo di destinazione
- **V:** La **velocità del canale** di trasmissione tra i nodi
- **T:** **Tempo di attesa massimo** per l'indirizzamento di un messaggio di risposta dal dispositivo (nodo) al collettore (master), che comprende anche il tempo impiegato dalla verifica di disponibilità della rete (es. preamble, ascolto, ecc.) ed altri eventuali tempi richiesti dal protocollo.

Il valore del **time-out** risultante può essere utilizzato per determinare se un messaggio ha raggiunto la sua destinazione o meno. Nell'ultimo caso (messaggio di risposta dal nodo non ricevuto), sarà il nodo successivo (specificato nel percorso d'istradamento) ad inviare un messaggio di errore al Master.

2.3 TABELLE

2.3.1 Noise Table (Slave)

Ogni nodo gestisce **localmente** una “**tabella di rumore**” dove vengono memorizzati tutti gli altri nodi ad esso visibile (raggiungibili tramite un messaggio senza errori) nonché il loro relativo valore di **rumore**. La suddetta informazione è organizzata tramite l'utilizzo di **tuple** del tipo **<nodo mittente, nodo destinatario, rumore, bit di controllo>**:

Nodo mittente	Nodo destinatario	Rumore	Bit di controllo
...

- **Nodo mittente:** Hop che ha inviato il messaggio su cui è stato rilevato il rumore
- **Nodo destinatario:** Nodo che ha ricevuto il messaggio e rilevato il rumore.
- **Rumore:** Valore del disturbo quantificato in base alla decodifica di Hamming, e classificato in due livelli assoluti:
 - **Trascutabile:** Qualità del segnale di trasmissione fosse ottimo, la connessione è considerata “**senza rumore**”.
 - **Considerabile:** Segnale talmente disturbato da rendere illeggibile il contenuto, la connessione è considerata “**assente**”.
- **Bit di controllo:** Un bit finale utilizzato per sapere quando una **tupla di rumore** è stata modificata. I possibili valori sono:
 - **0:** Tupla non modificata.
 - **1:** Tupla modificata recentemente.

2.3.2 Noise Table (Master)

Il **master** gestisce una propria e unica “**tabella di rumore**” dove memorizza tutti i dati ricevuti dai vari nodi tramite i servizi di “**Request Noise Table**” e “**Send Response**”, organizzando l'informazione in **tuple** del tipo **<nodo mittente, nodo destinatario, rumore, timestamp>**, ovvero:

Nodo mittente	Nodo destinatario	Rumore	Timestamp
...

I contenuti di questa tabella vengono utilizzata dal **Master** come parametri principali per l'**istradamento** dei pacchetti tramite il rispettivo algoritmo.

3. REQUISITI

3.1 Requisiti Funzionali

Requisiti riguardanti cosa deve e/o non deve fare il sistema così come la sua reazione agli stimoli esterni:

ID	DESCRIZIONE	MoSCoW
1.	Un unico dispositivo, detto Master , elabora sia l'informazione acquisita dai sensori che dei parametri di routing.	Must Have
2.	Molteplici dispositivi, detti Slave , contribuiscono al ricavo e misurazione della qualità del segnale come parametro di routing .	Must Have
3.	I pacchetti , oltre a trasportare informazione (payload e path-route) devono consentire, se necessario, l'identificazione e correzione di un eventuale deterioramento	Must Have
4.	Il trasporto dei pacchetti di “Forwarding” avviene sempre dal Master verso un nodo destinatario, attraversando i dispositivi (hop) che formano il percorso mappato nel pacchetto.	Must Have
5.	La qualità del segnale è misurata in termini del “rumore” riscontrato durante il trasporto della informazione tra nodi (in qualità di hop, destinatari o semplici ascoltatori).	Must Have
6.	Il valore del “rumore” tra nodi è memorizzato in una tabella detta “Noise table” come tuple del tipo <sorgente, destinatario, rumore>, aggiornata ad ogni ricezione di pacchetto attraverso il protocollo “Update Noise” .	Must Have
7.	La zona di “visibilità” di un nodo non è altro che il suo raggio di comunicazione con altri nodi, essa è determinata dalla noise table.	Must Have
8.	Quando un dispositivo (slave) si collega alla rete, si identifica inviando al Master un messaggio di protocollo “Hello” (col suo MAC Address) rimanendo in attesa di una conferma.	Must Have
9.	Il Master assegna un indirizzo di sottorete al nuovo nodo e gli conferma tramite il protocollo di “Request Noise Table” il suo ingresso nella rete richiedendo la sua “Noise table” locale.	Must Have
10.	Il controllo dei dispositivi (rilevatori, attuatori, ecc.) è eseguito dal Master tramite l'invio di un messaggio al relativo gestore (Slave) utilizzando il protocollo “Send Response” .	Must Have
11.	Nel caso un nodo non riceva nessun messaggio durante un certo lasso di tempo, il protocollo di “Watchdog” procederà al rinvio di un messaggio di “Hello” .	Must Have

3.2 Requisiti Non-Funzionali

Requisiti che definiscono le proprietà del sistema che devono essere soddisfatte:

ID	DESCRIZIONE	MoSCoW
1.	Il Master deve essere basato su un dispositivo Arduino	Must Have
2.	Ogni Slave deve essere basato su un dispositivo Raspberry	Must Have
3.	La velocità di risposta degli Slave ai comandi del Master deve essere veloce	Should Have
4.	Il numero di Slave da gestire può essere illimitato	Could Have
5.	Il software implementato nel Master e Slave deve essere sviluppato in C	Won't Have

4. ANALISI

In questa sezione verranno presentati i diagrammi dello standard **UML** (Unified Modeling Language), con lo scopo di agevolare e dare sostegno in fase di analisi sia agli analisti che ai clienti.

4.1 GELLISH

Il dizionario presentato in questo capitolo è stato elaborato con **Gellish**, lo strumento utilizzato per eliminare le ambiguità del linguaggio specialistico utilizzato nel dominio del sistema. Il dizionario risultante è basato sul linguaggio naturale dove sono stati incluse tutte le **definizioni** riguardanti il sistema, tramite l'utilizzo di termini e relazioni standard presenti nel dizionario Gellish.

COMUNICATION PROTOCOL FOR UTILITY NETWORKS												
Gellish	Version:	9.0	19/11/2017									65
0	54	71	16	101	1	60	3	15	45	201		
1	Language of left hand object name	ID of Context for left hand object name	Name of Context for left hand object name	Left hand object name	Fact id	Relation type ID	Relation type name	Unique id of right hand object	Simultaneous right hand cardinalities	Right hand object name		Definition
2	English	492,015	Gellish English	Device	1,0 08, 563	1,981	is a synonym of	1.313	0,1	connected member	E' una unità hardware come ad esempio una periferica di un computer o un dispositivo elettronico. Si dice in particolare di dispositivi e apparecchi ad alta tecnologia e di piccole dimensioni. Sta sempre in ascolto, e aggiorna di continuo la propria noise table se la visibilità verso gli altri nodi vicini lo permette	
3	English	1.300	Powerline smartgrid with Master/Slave model	Master	1,0 08, 563	1,981	is a specialization of	492,015		device	Il master è un tipo di device in una network di tipo master/slave composta da devices. Il master si occupa generalmente della gestione dei protocolli di controllo e funzionamento che mantengono funzionale e operativa la rete stessa.	
4	English	1.300	Powerline smartgrid with Master/Slave model	Slave	1,0 08, 563	1,981	is a specialization of	492,015		device	Lo Slave è un tipo di device in una network di tipo master/slave composta da un certo numero di devices. Lo slave è un dispositivo che subisce come nel nostro caso una qualche funzione privilegiata di controllo o di concentrazione dati da parte del dispositivo Master ma può comunque essere dotato di un'autonomia di controllo o di memorizzazione di dati.	
5	English	1.300	Powerline smartgrid with Master/Slave model	Visibilità	1.0 06. 37 6	5.091	can be a realization of a	492,015		device	Capacità di un nodo di rilevare altri nodi presenti nel suo raggio di comunicazione e di visione in una certa porzione di network ove ha capacità di ricevere e analizzare distintamente i pacchetti che a lui giungono.	

Communication protocol for Utility networks

6	English	1.303	Powerline smartgrid with Master/Slave model	Network	1,00 8,01 63	5.229	has conceptually a role as	4.499	0,1	Network Poweline, Network Master/Slave	Una Network nel senso più generico è intesa come un insieme di dispositivi hardware e software collegati l'uno con l'altro da appositi canali di comunicazione, che permette il passaggio da un dispositivo all'altro di risorse, informazioni e dati in grado di essere elaborati e condivisi.
7	English	1.303	Powerline smartgrid with Master/Slave model	Network Master/Slave	1,0 08, 245 3	1,146	is a specialization of	1.303		Network	Una rete Master/slave implementa un modello di comunicazione in cui un dispositivo o processo ha un controllo unidirezionale su uno o più altri dispositivi. In questi sistemi viene selezionato un master da un gruppo di dispositivi idonei, con gli altri dispositivi che agiscono nel ruolo di slave
8	English	1.303	Powerline smartgrid with Master/Slave model	Network Powerline grid	1,0 08, 215 3	1,146	is a type of	1.303		Network	il powerline (PLC) è una tecnologia per la trasmissione di voce o dati che utilizza la rete di alimentazione elettrica come mezzo trasmittivo. Si realizza sovrapponendo al trasporto di corrente elettrica, continua o alternata a bassa frequenza un segnale a frequenza più elevata che è modulato dall'informazione da trasmettere. La separazione dei due tipi di correnti si effettua grazie al filtraggio e separazione degli intervalli di frequenze utilizzate.
9	English	492,015	Gellish English	Node	1,0 08, 215 3	1,149	is a synonym of	492,015		Device	Dispositivo facente parte dal sistema Master o Slave che sia. E' un modo per intendere un qualsiasi device facente parte della Network Master/Slave(dispositivi già identificati). In un ottica di tipo astratto e progettuale sono la rappresentazione dei dispositivi devices che fanno parte e di cui la rete si compone, ove la rete è un grafo e gli archi rappresentano le linee di comunicazione e i nodi rappresentano i devices, cioè genericamente dispositivi emittenti e riceventi.
10	English	1.303	Powerline smartgrid with Master/Slave model	Terminal	1,0 08, 215 3	1,146	is a specialization of	492,978		slave	Nodo slave che è destinatario di un messaggio inoltrato dai servizi disponibili del protocollo e proveniente dal Master. Una volta che il suddetto nodo terminal ha eseguito i compiti specificati e/o direttamente codificati dalla composizione dei pacchetti ove il messaggio viaggia in rete allora il nodo terminal reinvia opportunamente la risposta al master
11	English	1.303	Powerline smartgrid with Master/Slave model	Hop	1,0 08, 215 3	1,146	is a specialization of	492,978		slave	Nodo slave che fa da ponte tra il Master ed un Terminal effettuando il forward dei messaggi. In pratica è un nodo transitario che reindirizza in base al percorso di routing mappato nel messaggio stesso il medesimo verso l'opportuna uscita nella zona di rete ove è in ascolto il successivo hop o direttamente il terminal.
12	English	1.419	Powerline smartgrid with Master/Slave model	Forward	1,0 08, 215 3	1,153	is used by	1.419		Protocols	Inoltro di un messaggio e quindi di una serie di pacchetti in arrivo verso un altro hop. Cioè un altro device, che può essere il master nel caso il messaggio sia giunto al terminal e sta compiendo a ritroso la strada verso il master oppure che va verso il terminal.
13	English	1.419	Powerline smartgrid with Master/Slave model	Protocols	1,0 08, 215 3	1,153	is used by	492,015		devices	L'insieme delle regole e delle procedure che governano lo scambio di informazioni e dati tra i devices nella network powerline. Si basano sulla convergenza di opportuni algoritmi, sulla struttura dei pacchetti e sui dati contenuti nelle noise table.
14	English	1.419	Powerline smartgrid with Master/Slave model	Algorithms	1,0 08, 215 3	1,157	is a generalization of	1.233		algoritmi di instadramento	Un algoritmo è un procedimento che risolve un determinato problema attraverso un numero finito di passi elementari.

Communication protocol for Utility networks

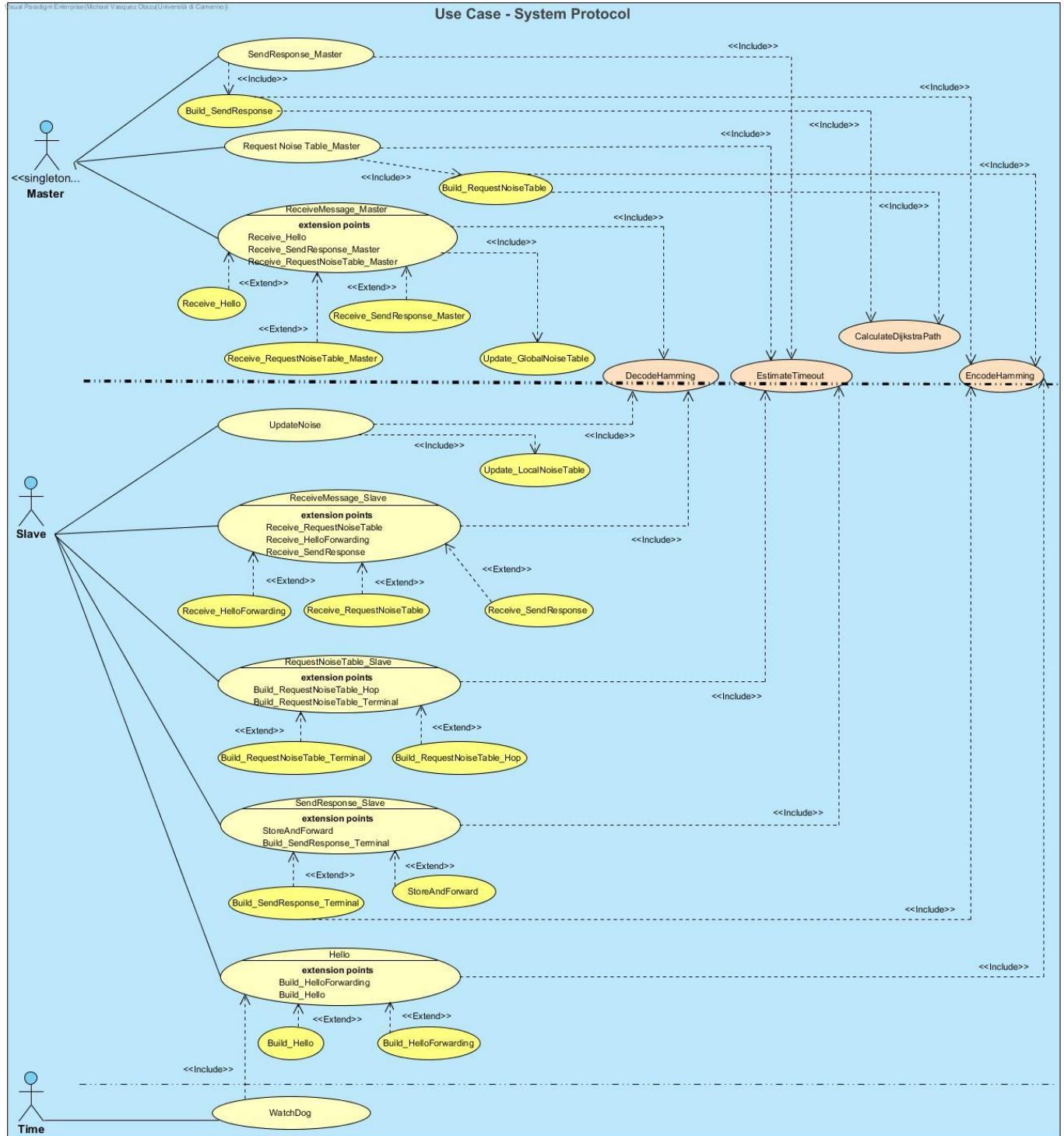
15	English	1.419	Powerline smartgrid with Master/Slave model	Algorithmi di instadramento	1,0 08, 215 3	1,146	is a specialization of	1.233		algorithm	L'algoritmo di instradamento (o routing) è in sostanza uno dei protocolli che gestisce il corretto funzionamento della network poiché permette il corretto instradamento dei pacchetti verso la giusta destinazione.
16	English	1.419	Powerline smartgrid with Master/Slave model	Servizio del protocollo: Hello	1,0 08, 215 3	1,146	is a specialization of	1.419		Protocols	E' un servizio del protocollo di rete che permette ai dispositivi Slave appena collegati alla Network Powerline di comunicare al master (e di conseguenza a tutti i nodi della rete) la loro presenza. Questo perché il Master deve essere capace di tracciare e veicolare ogni nodo non appena si collega alla rete e quindi tramite la seguente procedura può fare una topologia di tutta la Network.
17	English	1.419	Powerline smartgrid with Master/Slave model	Servizio del protocollo: Update Noise	1,0 08, 215 3	1,146	is a specialization of	1.419		Protocols	Il valore del "rumore" tra nodi è memorizzato in una tabella detta "Noise table" aggiornata ad ogni ricezione di pacchetto attraverso tale servizio del protocollo. Il servizio in pratica consiste che ogni nodo effettua un continuo rilevamento dei valori del rumore nei pacchetti in transito nella sua zona di visibilità decodificandoli e rilevando poi gli errori eventualmente presenti.
18	English	1.419	Powerline smartgrid with Master/Slave model	Servizio del protocollo: Request Noise Table	1,0 08, 215 3	1,146	is a specialization of	1.419		Protocols	Quando il Master riceve un messaggio di "Hello" proveniente da un nuovo dispositivo (slave), esso comunica al medesimo il suo ingresso nella rete attraverso il servizio di "Request Noise Table". Tale consiste nell'invio di un messaggio contenente la richiesta della sua "Noise Table" locale, che servirà al Master per stabilire un primo percorso di routing (attraverso i nodi adiacenti) basato sul rumore percepito dal nuovo dispositivo
19	English	1.419	Powerline smartgrid with Master/Slave model	Servizio del protocollo: Send Response	1,0 08, 215 3	1,146	is a specialization of	1.419		Protocols	Il controllo dei dispositivi (rilevatori, attuatori, ecc.) è eseguito dal Master tramite l'invio di un messaggio al relativo nodo (Slave) utilizzando il servizio di Send Response, che permette appunto di veicolare ogni nodo presente richiedendo informazioni sullo stato del dispositivo elettronico che sta controllando oppure nell'operare determinate funzioni verso di esso.
20	English	1.419	Powerline smartgrid with Master/Slave model	Servizio del protocollo: Watchdog	1,0 08, 215 3	1,146	is a specialization of	1.419		Protocols	Il servizio Watchdog permette ai nodi di verificare e/o rinnovare la propria presenza nella rete. Cioè, nel caso un nodo non riceva nessun messaggio durante un certo lasso di tempo, il protocollo di Watchdog procederà al reset completo del device e al ripristino con relativo rinvio di un messaggio di Hello.
21	English	1.419	Powerline smartgrid with Master/Slave model	Rumore(Noise)	1,0 08, 215 3	1,132	is a problem	1.303		Network Powerline grid	In elettronica il rumore è l'insieme di segnali, in tensione o corrente elettrica, imprevisti e indesiderati che si sovrappongono al segnale utile, trasmesso o da elaborare, tipicamente presente sul canale di comunicazione tra due o più dispositivi di ricezione/elaborazione. Il rumore provoca una perdita d'informazione o un'alterazione del messaggio trasmesso.

Communication protocol for Utility networks

22	English	1.419	Powerline smartgrid with Master/Slave model	Pacco hetto	1,0 08, 215 3	1,146	is used by	1.303		Network Powerline grid	Viene detto pacchetto ciascuna sequenza finita e distinta di dati trasmessa su una rete. Tipicamente si tratta di una sequenza di bit, ovvero informazione in formato digitale, modulata poi in maniera numerica per la trasmissione sul canale fisico che nel nostro caso si tratta di una rete in powerline
23	English	1.419	Powerline smartgrid with Master/Slave model	Local Noise Table	1,0 08, 215 3	1,146	is used by	1.419		protocols	Tabella dove ogni nodo memorizza tutti i suoi nodi visibili assieme al loro rumore rilevato
24	English	1.419	Powerline smartgrid with Master/Slave model	Global Noise Table	1,0 08, 215 3	1,146	is used by	1.419		protocols	Tabella appartenente al master dove vengono caricate continuamente le ultime tuple modificate e quindi più aggiornate, provenienti dalle local noise table di tutti i nodi. Viene fornita grazie ai servizi del protocolli di funzionamento e permette la costruzione del percorso di routing poiché fornisce i giusti parametri all'algoritmo di instradamento
25	English	1.419	Powerline smartgrid with Master/Slave model	Tuples	1,0 08, 215 3	1,163	is a component of	22, 23		Local Noise Table, Global Noise Table	Si può definire una tupla su un insieme di attributi (o campi dato) X, come una funzione che a ciascun attributo appartenente ad X associa un valore appartenente al dominio dell'attributo. Si tratta dunque in sostanza di una riga di una tabella o relazione cioè l'insieme dei valori assunti dai campi dato o attributi specificati in cima a ciascuna colonna.

4.2 USE CASE

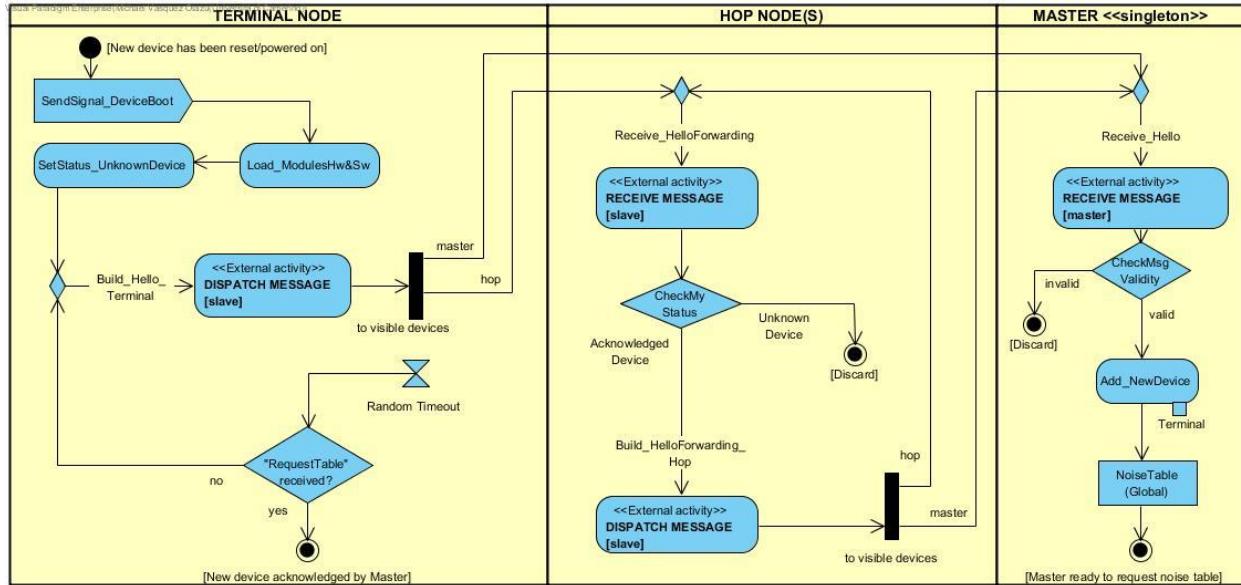
Gli **Use Case Diagram** (UCD) sono diagrammi dedicati alla descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso.



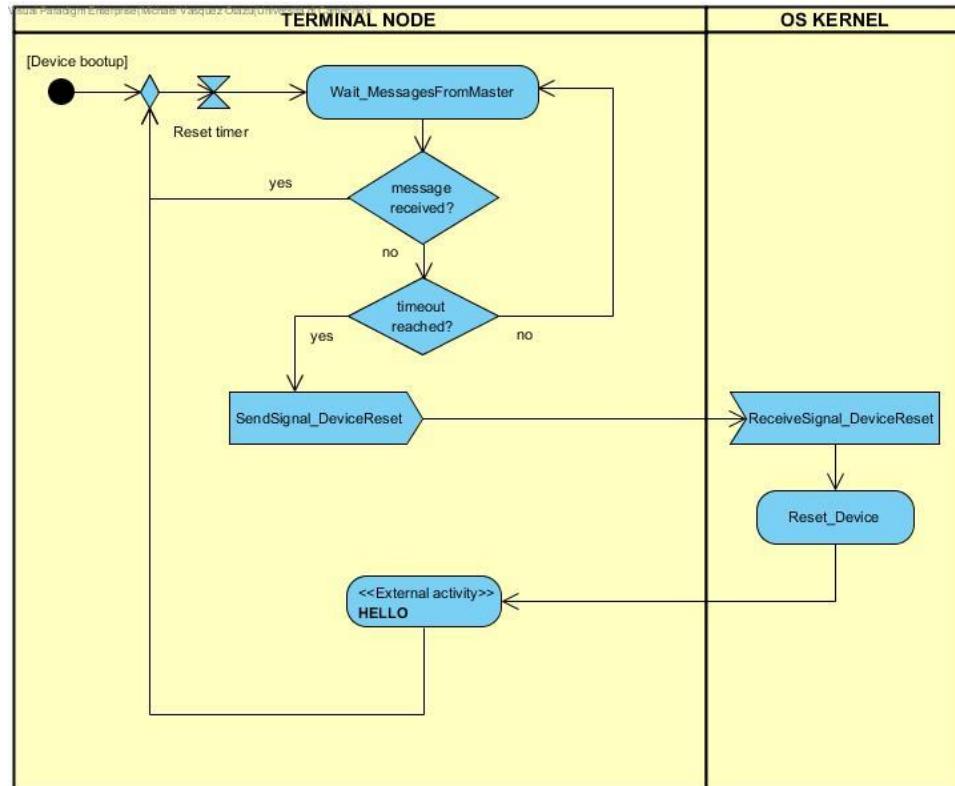
4.3 ACTIVITY DIAGRAM

Gli **Activity Diagram** (AD) presentati a continuazione hanno lo scopo di dettagliare i vari processi coinvolti nel sistema, definiti tramite una serie di attività o flussi. La sua lettura va effettuata ponendo un **token** nel nodo iniziale e seguendo il suo flusso fra gli altri nodi fino a giungere in un nodo finale.

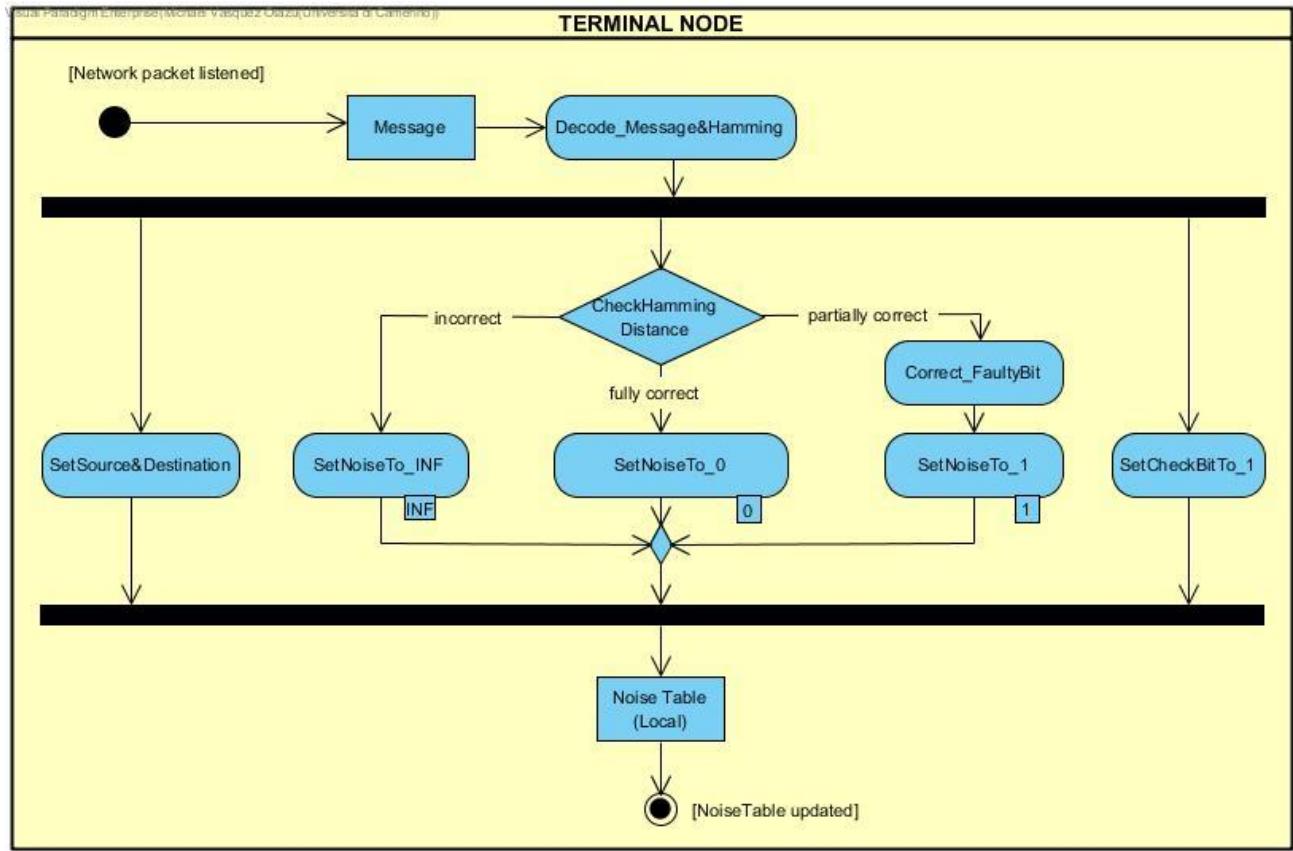
4.3.1 AD Hello



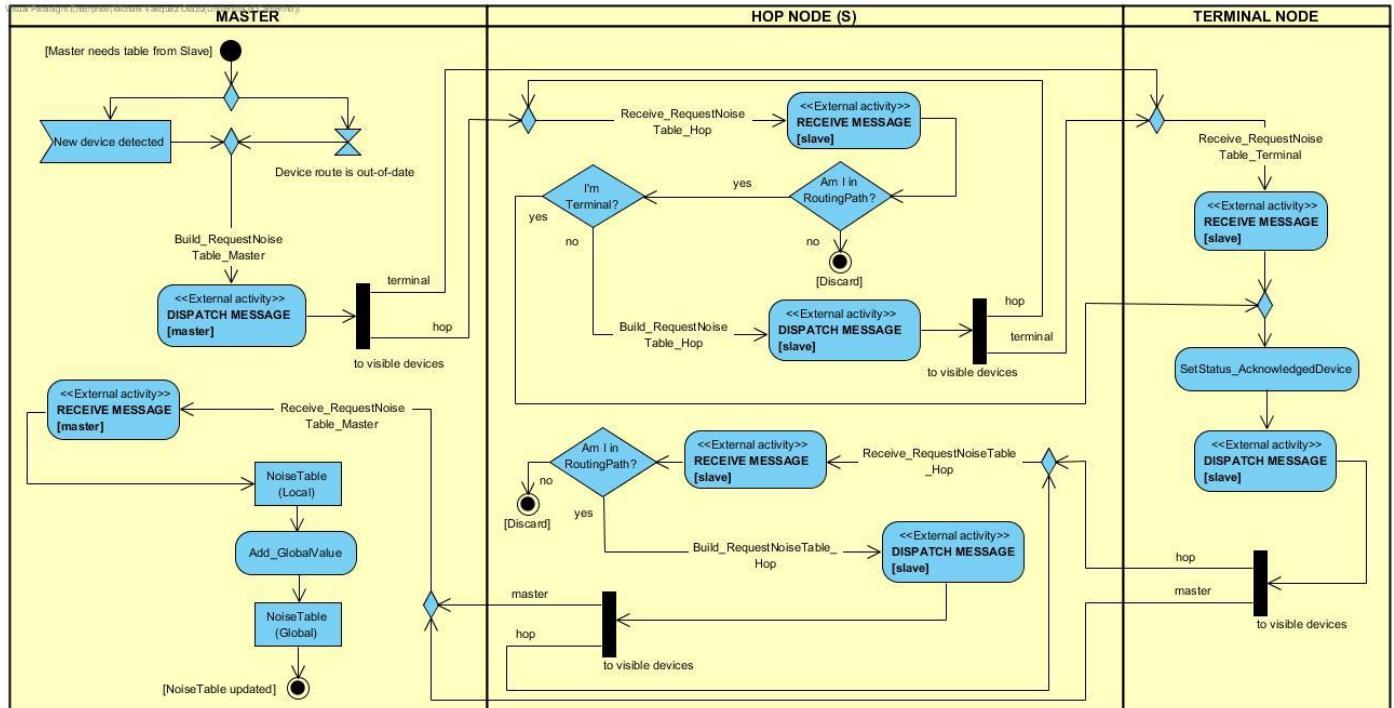
4.3.2 AD Watchdog



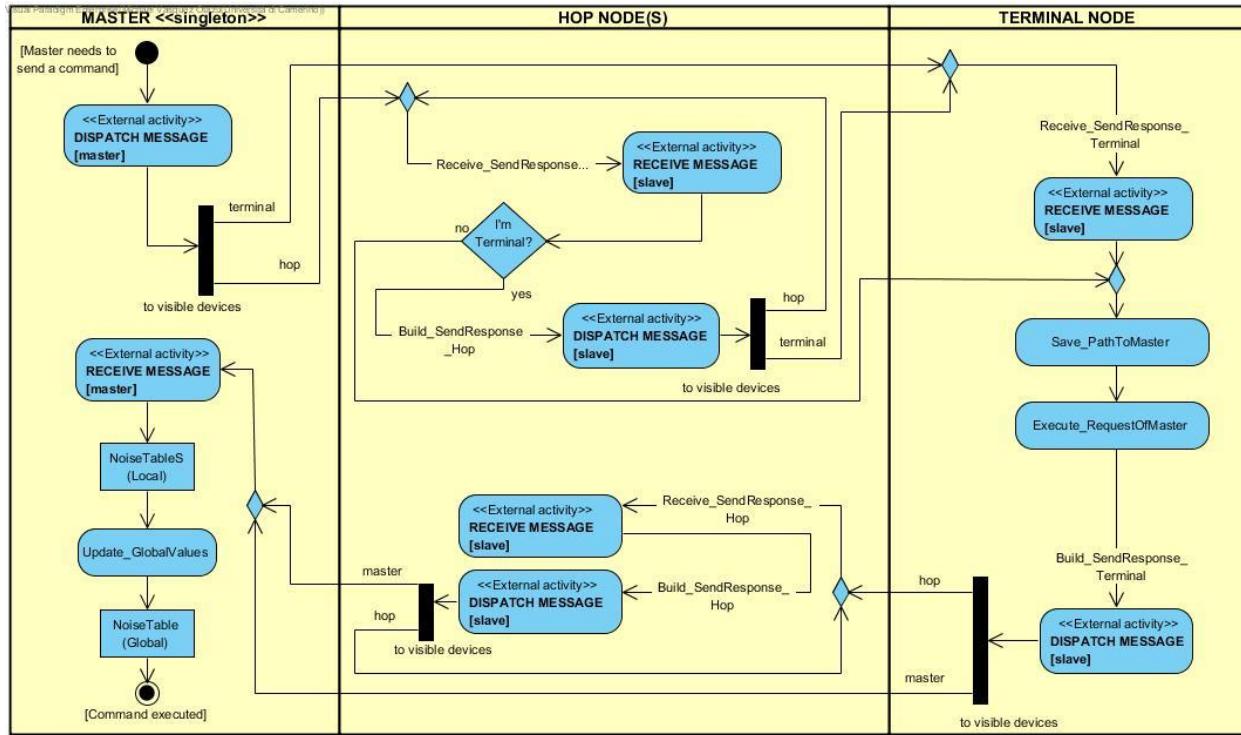
4.3.3 SD Update Noise



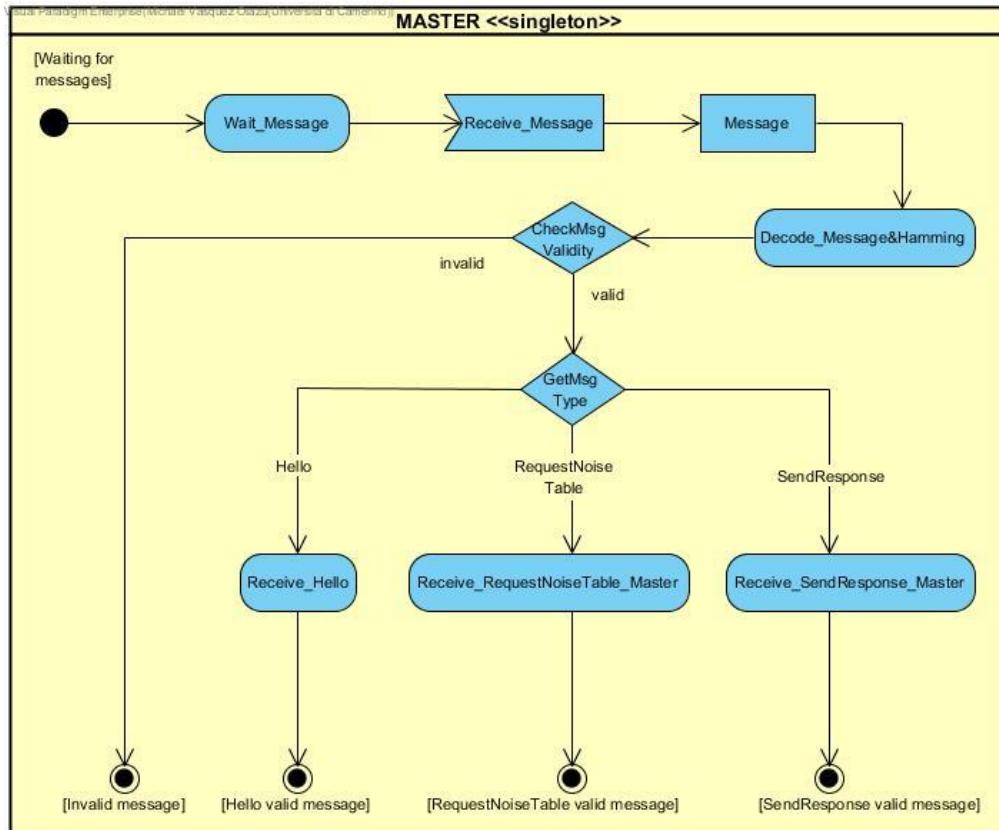
4.3.4 SD Request Noise Table



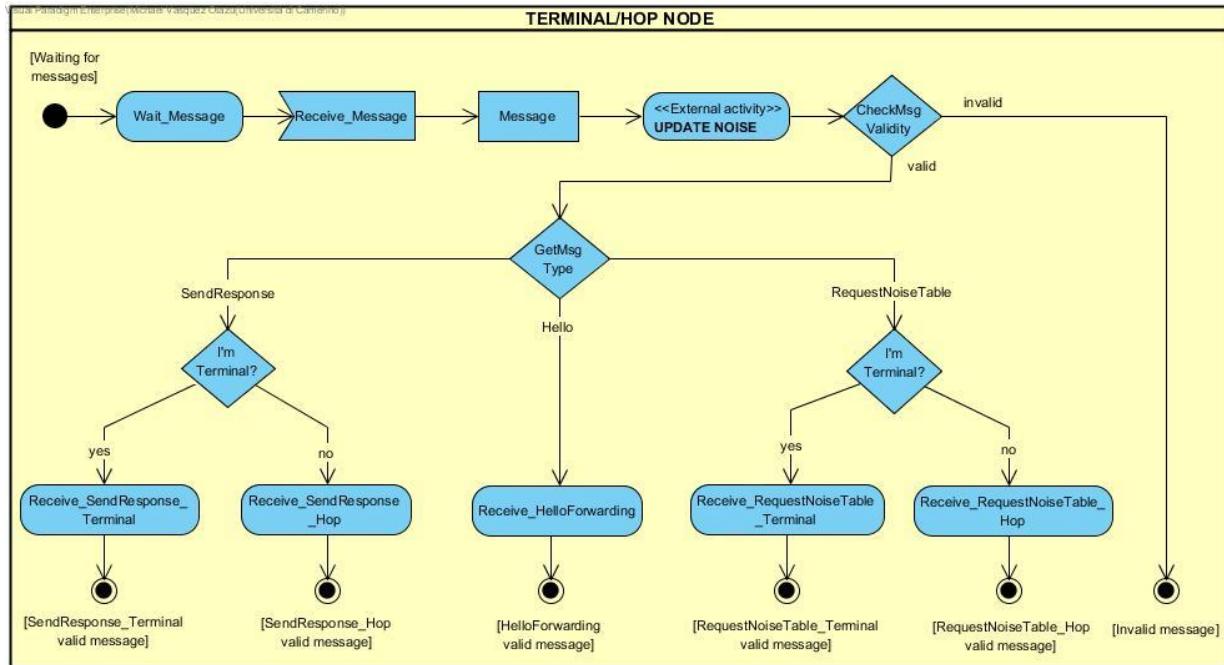
4.3.5 SD Send/Response



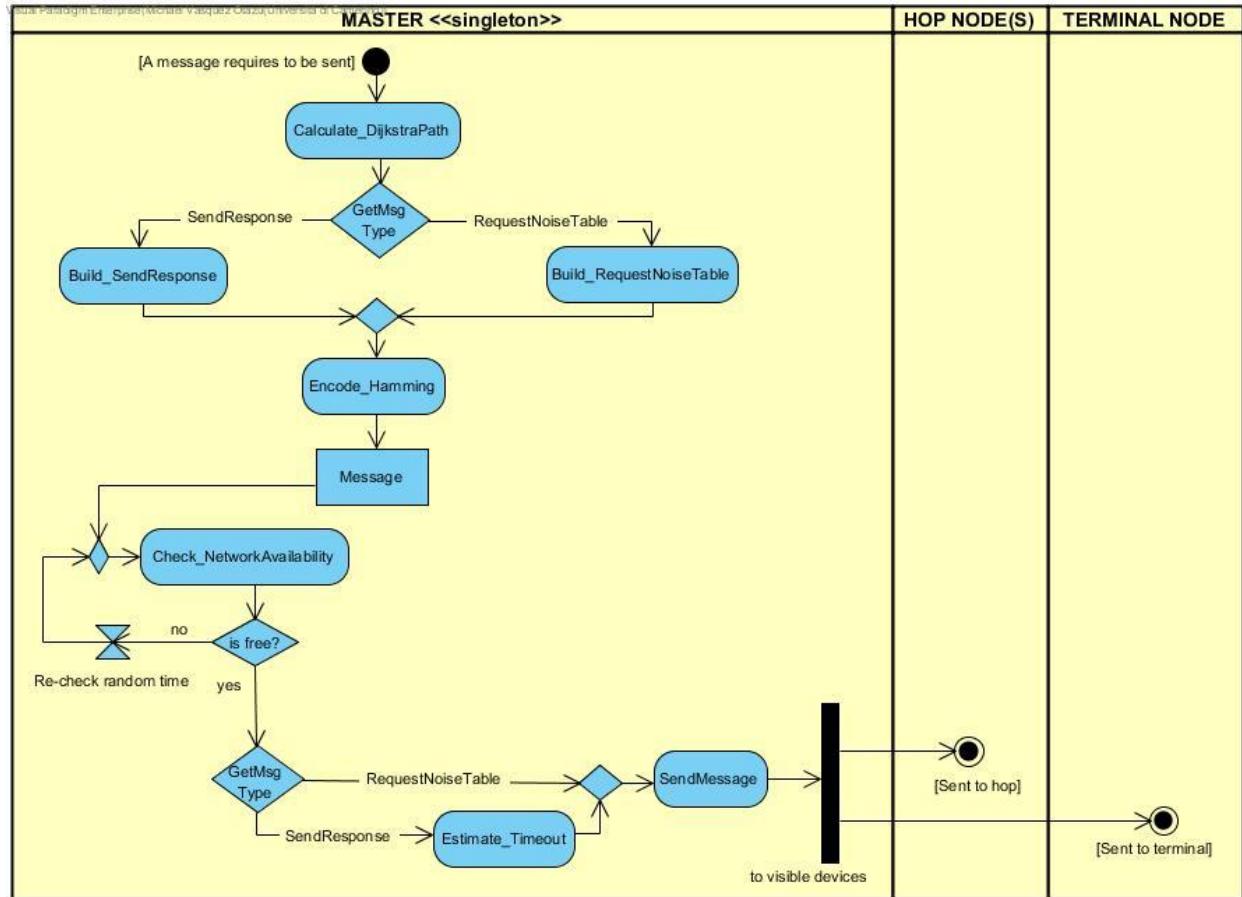
4.3.6 SD Receive Message (Master)



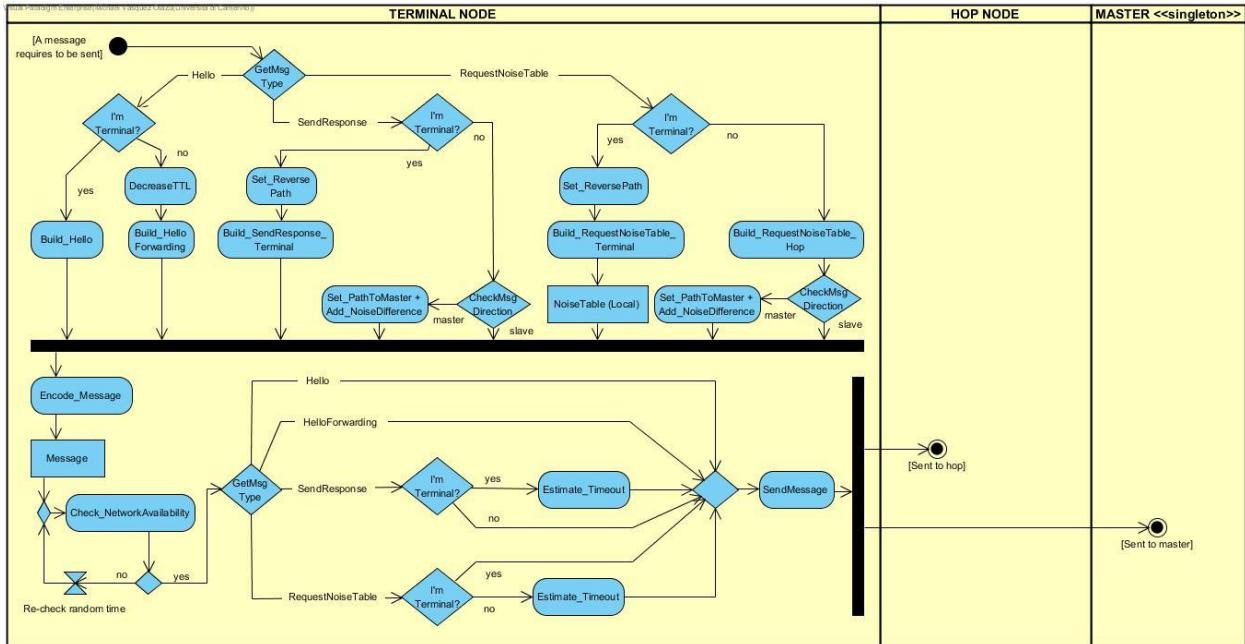
4.3.7 SD Receive Message (Slave)



4.3.8 SD Dispatch Message (Master)



4.3.9 SD Dispatch Message (Slave)

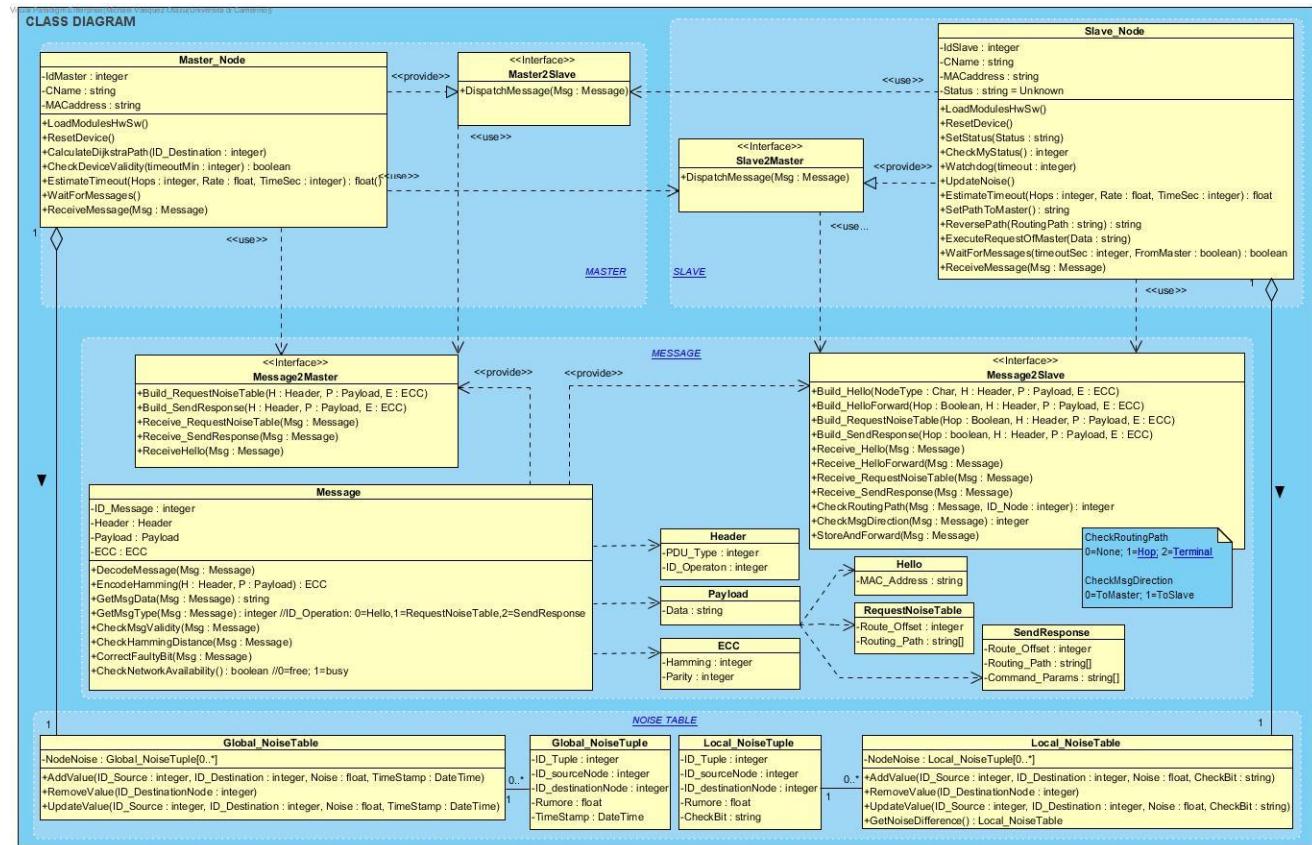


5. PROGETTAZIONE

Per passare dalla comprensione della richiesta alla fase di produzione si ricorre all'utilizzo dello **standard UML**, per espandere il materiale prodotto nelle fasi precedenti con quello specifico della progettazione.

5.1 CLASS DIAGRAM

I **Class Diagram** (CD) presentati in questa sezione consentono di descrivere i tipi di entità nel sistema, includendo le loro caratteristiche, funzioni ed eventuali relazioni fra loro. Nel nostro **CD** si possono identificare le seguenti entità principali:



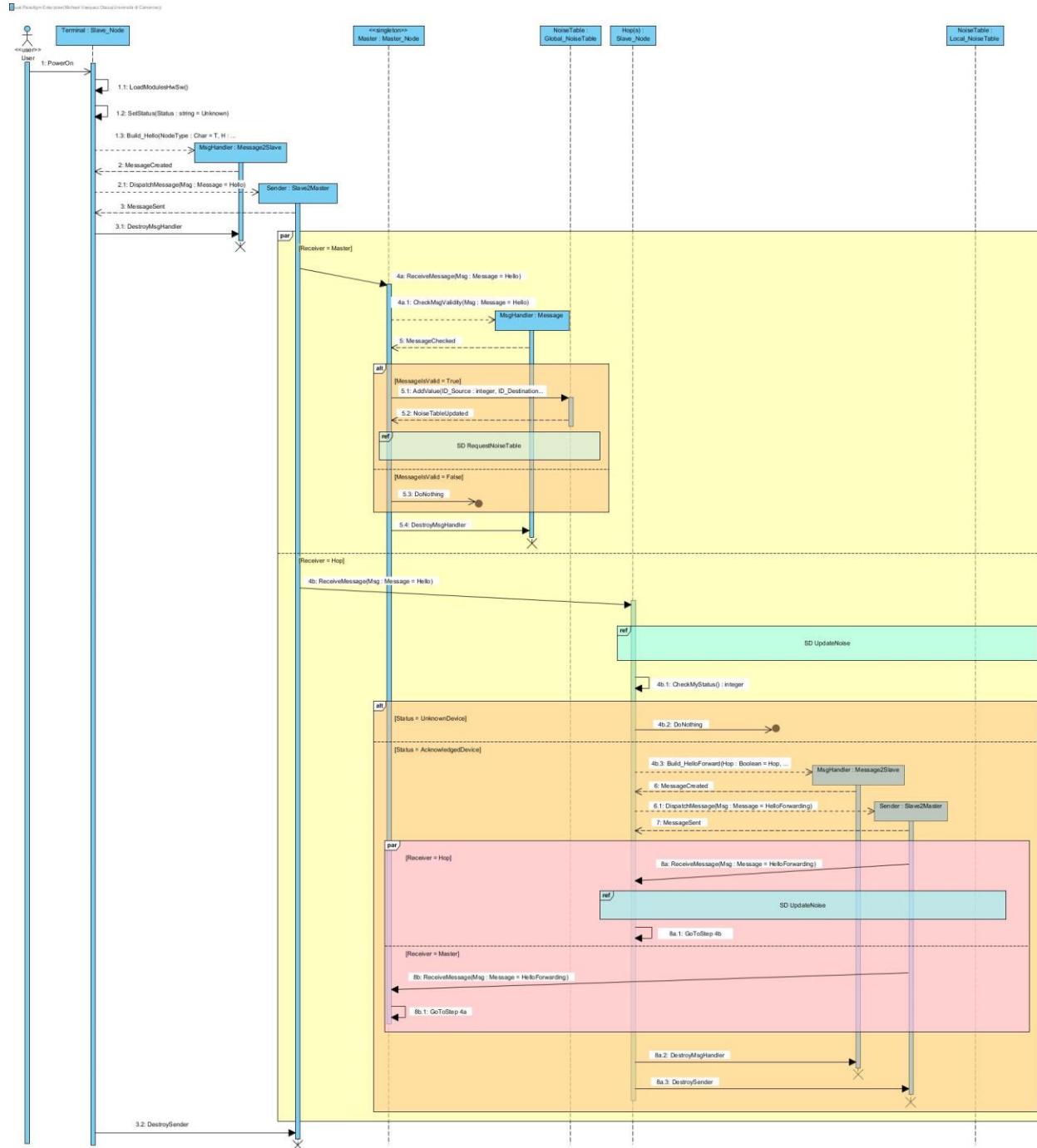
- **Master_Node**: Dispositivo collettore e di controllo
- **Slave_Node**: Dispositivo da controllare
- **Message**: Creazione e gestione dei messaggi
- **Global Noise Table**: Tabella globale di rumore del Master
- **Local Noise Table**: Tabella locale di rumore per ogni Slave

- **Master2Slave**: Metodi d'interfaccia dallo Slave verso il Master
- **Slave2Master**: Metodi d'interfaccia dal Master verso lo Slave
- **Message2Master**: Metodi d'interfaccia tra Messaggio e Master
- **Message2Slave**: Metodi d'interfaccia tra Messaggi e Slave

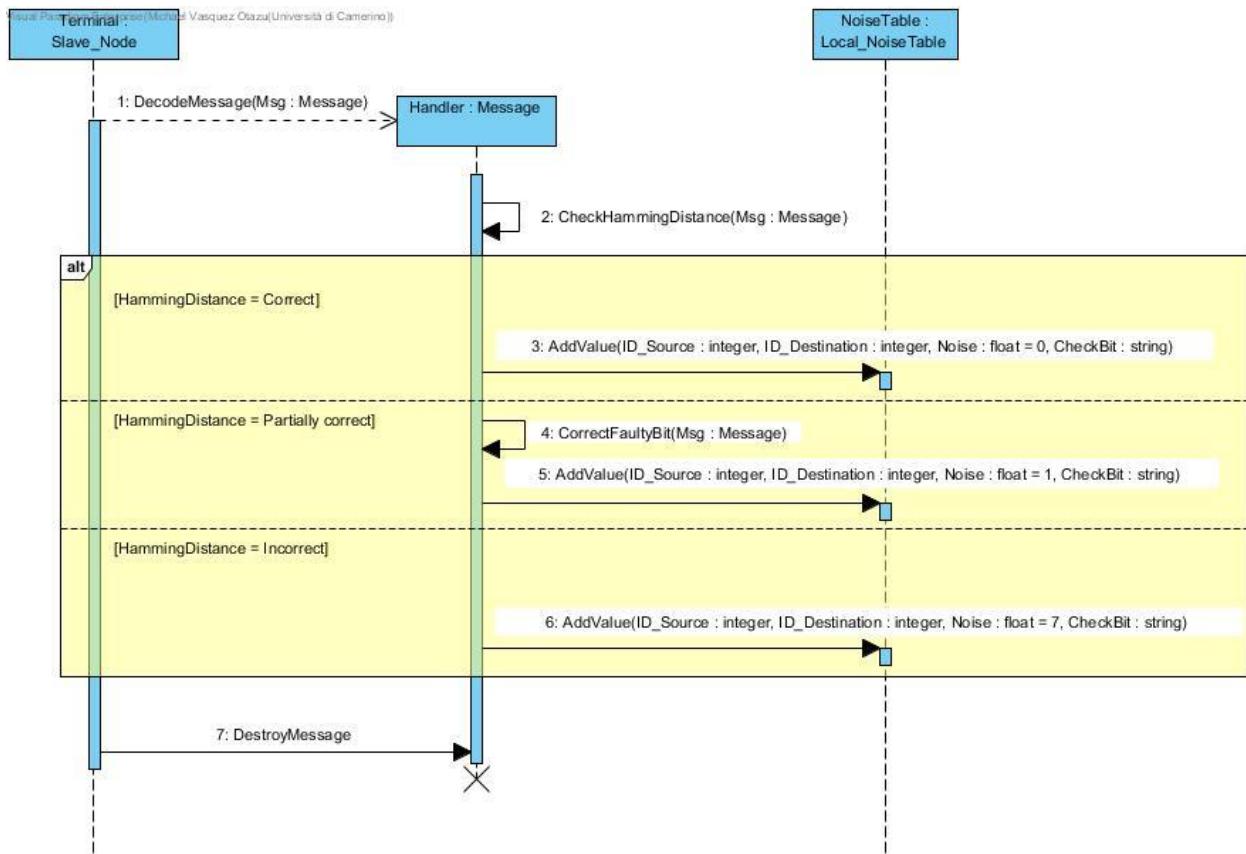
5.2 SEQUENCE DIAGRAM

I **Sequence Diagram (SD)** rappresentano tutti quei flussi già esaminati negli **Activity Diagram** ma in maniera molto più approfondita e più vicina a quella che sarà poi la reale stesura dell'algoritmo. Inoltre questo diagramma offre una visione di come differenti oggetti collaborino all'interno di uno stesso flusso per il raggiungimento di uno scopo comune.

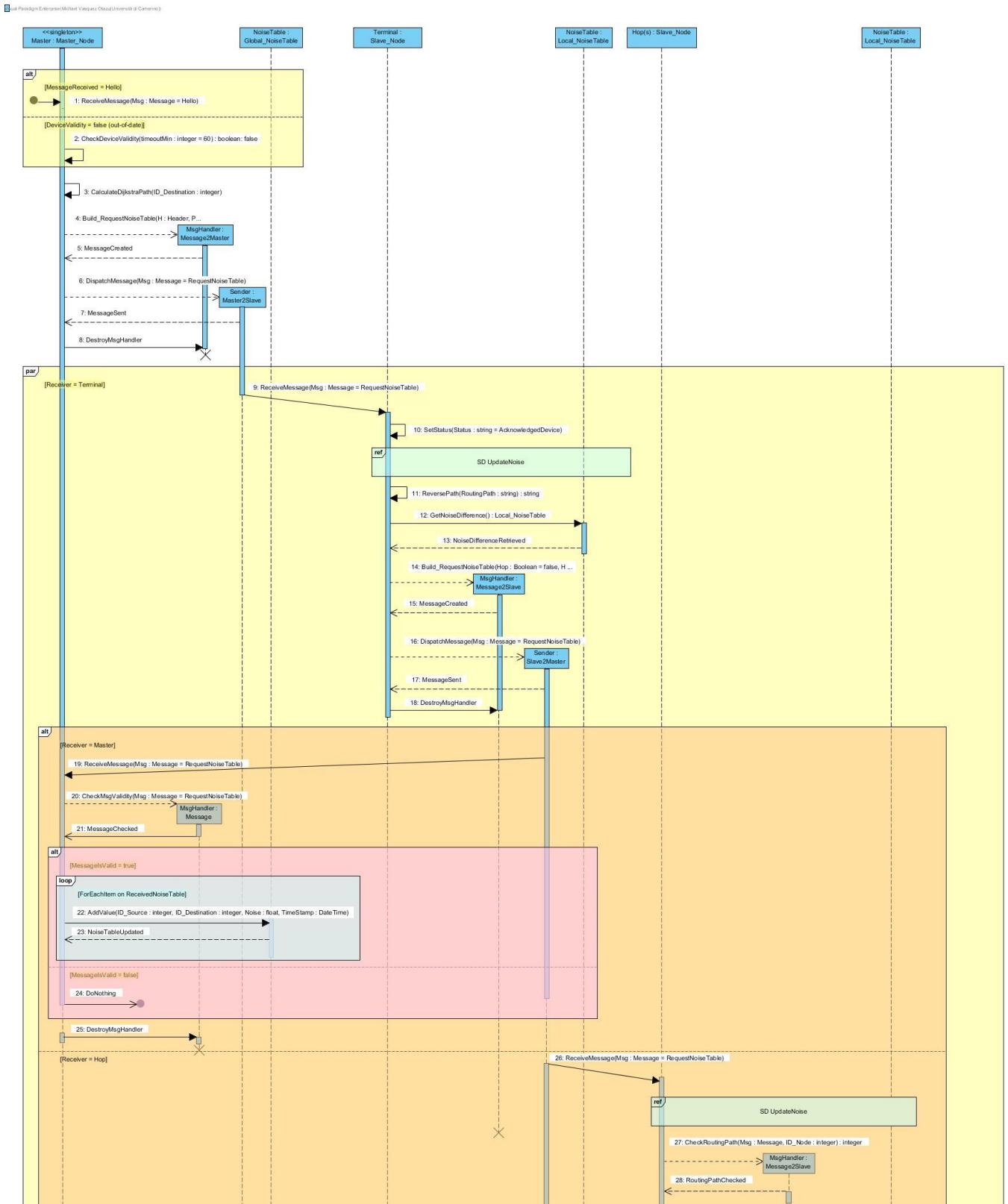
5.2.1 SD Hello



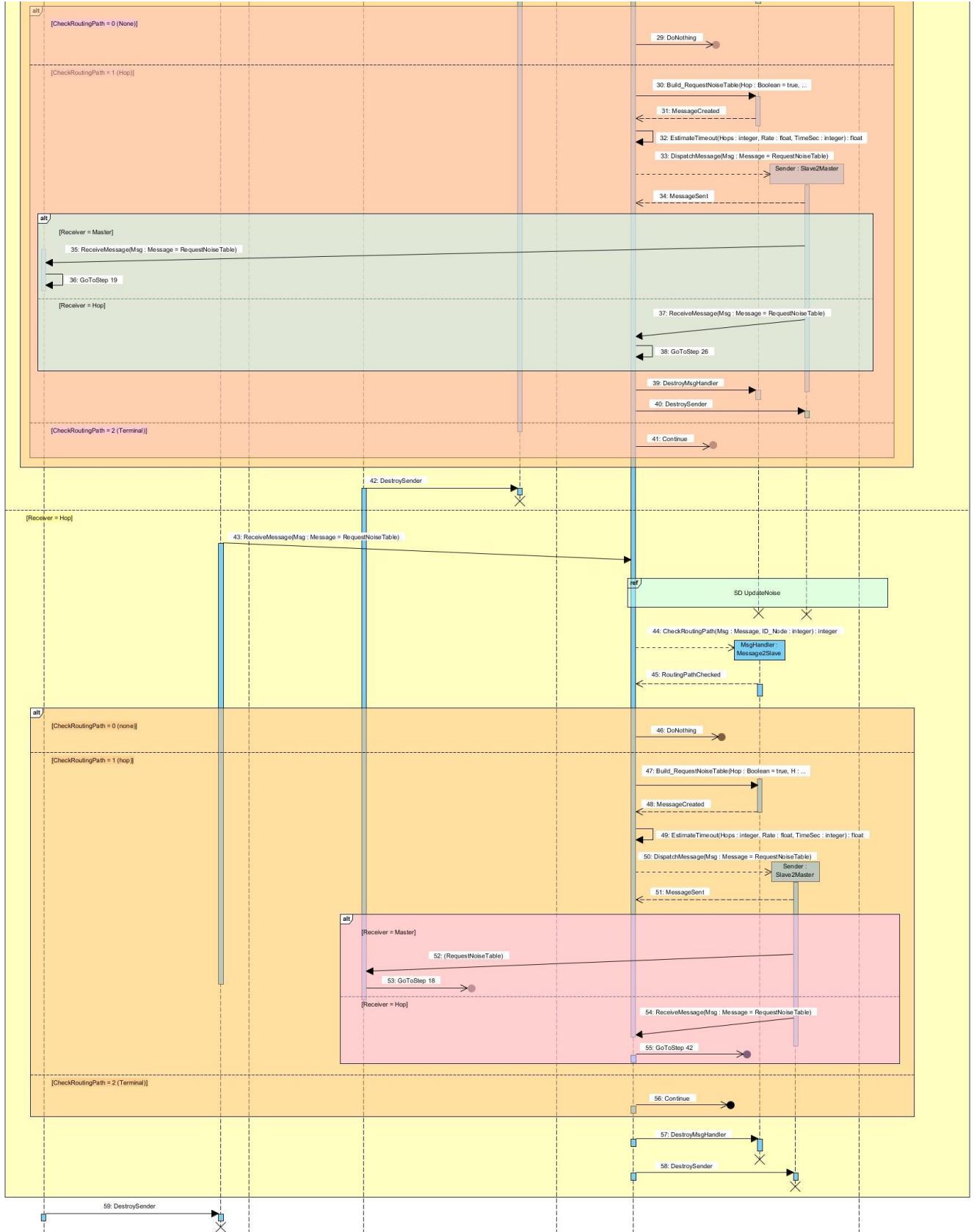
5.2.2 SD Update Noise



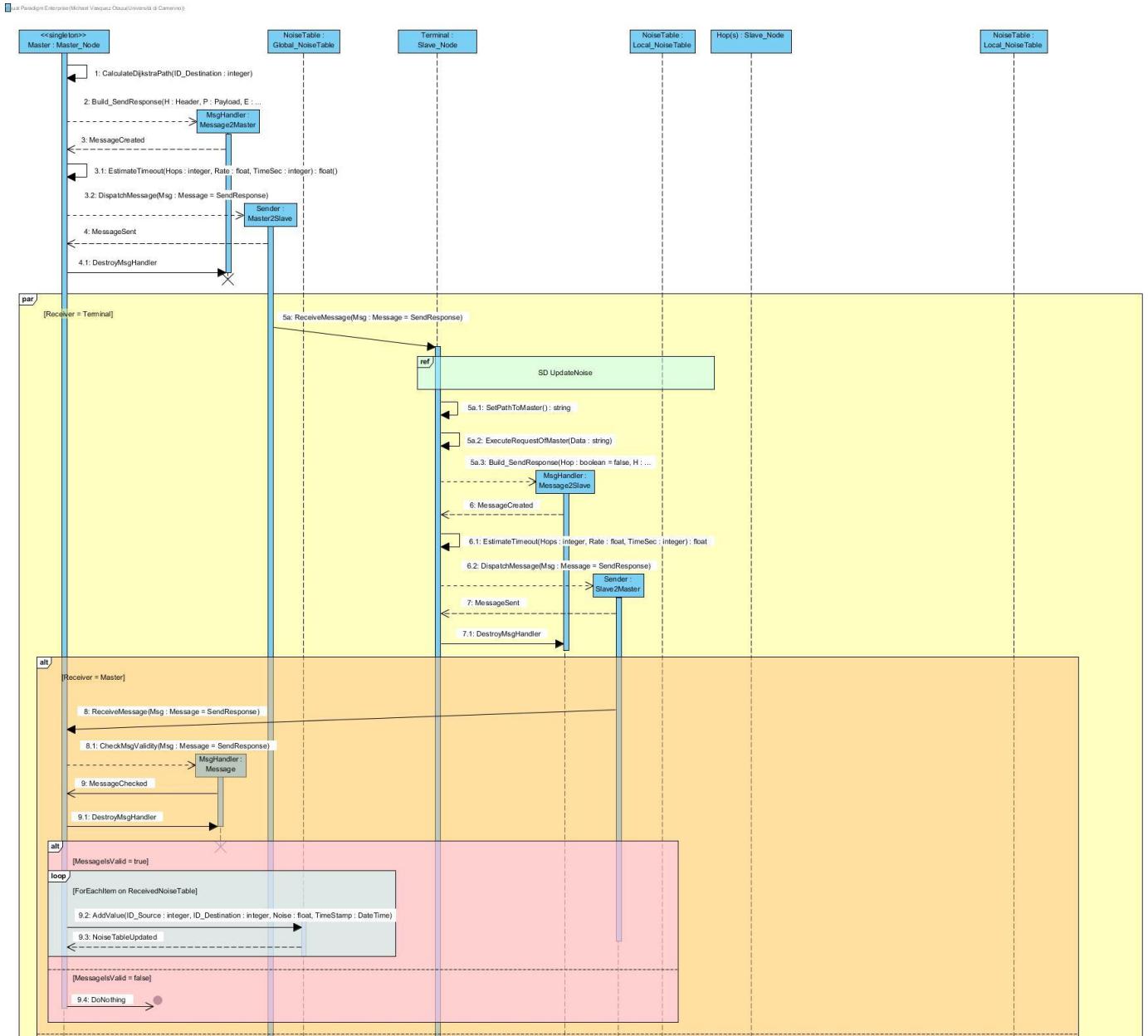
5.2.3 SD Request Noise Table



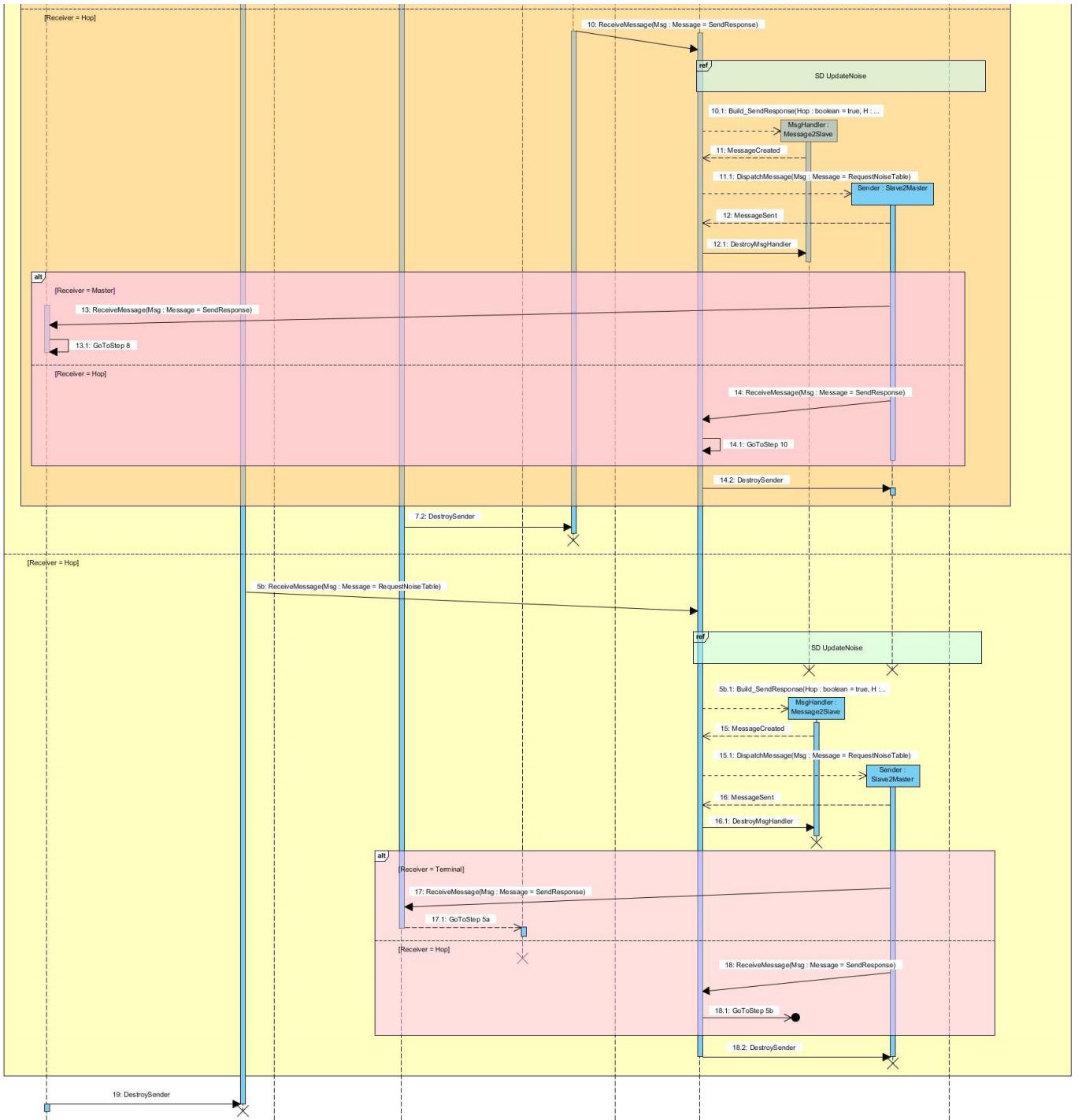
Communication protocol for Utility networks



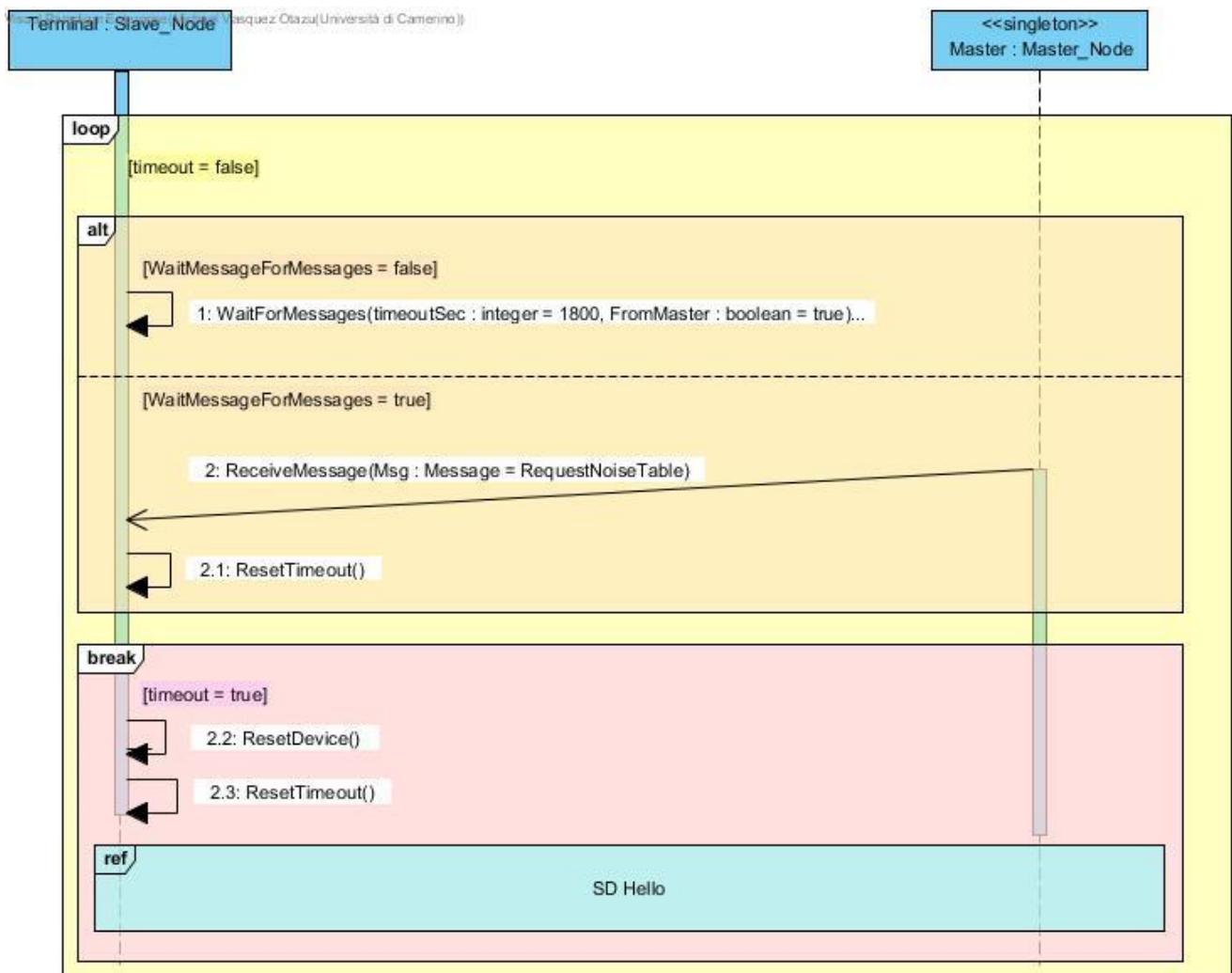
5.2.4 SD Send/Response



Communication protocol for Utility networks



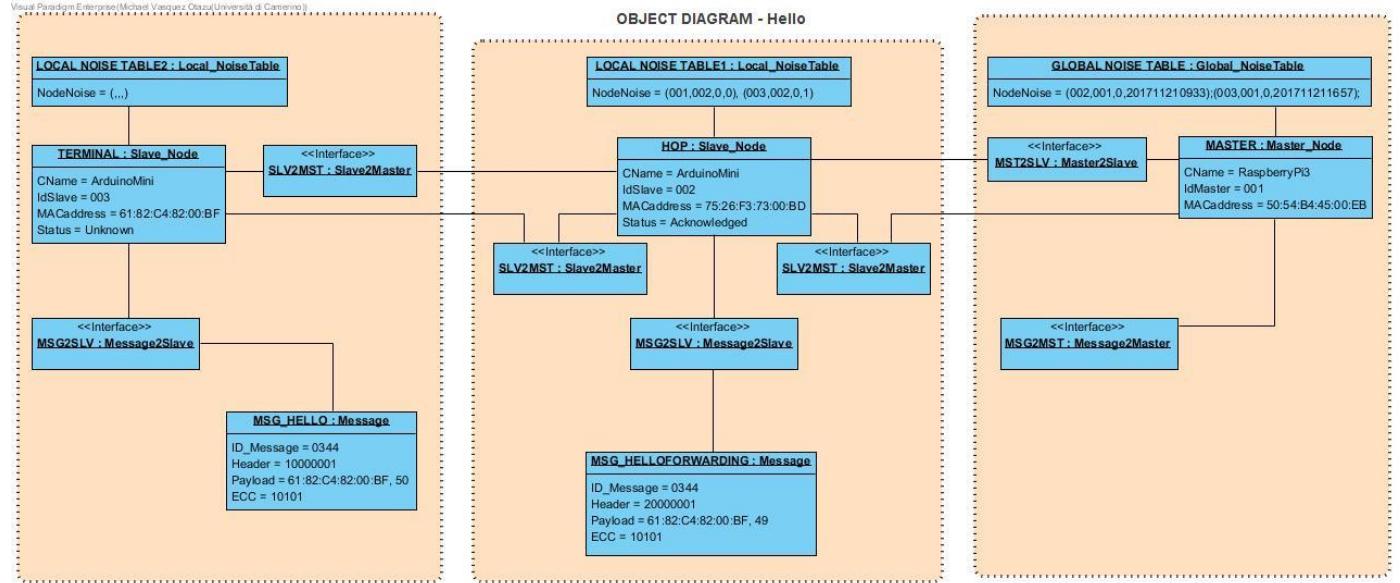
5.2.5 SD WatchDog



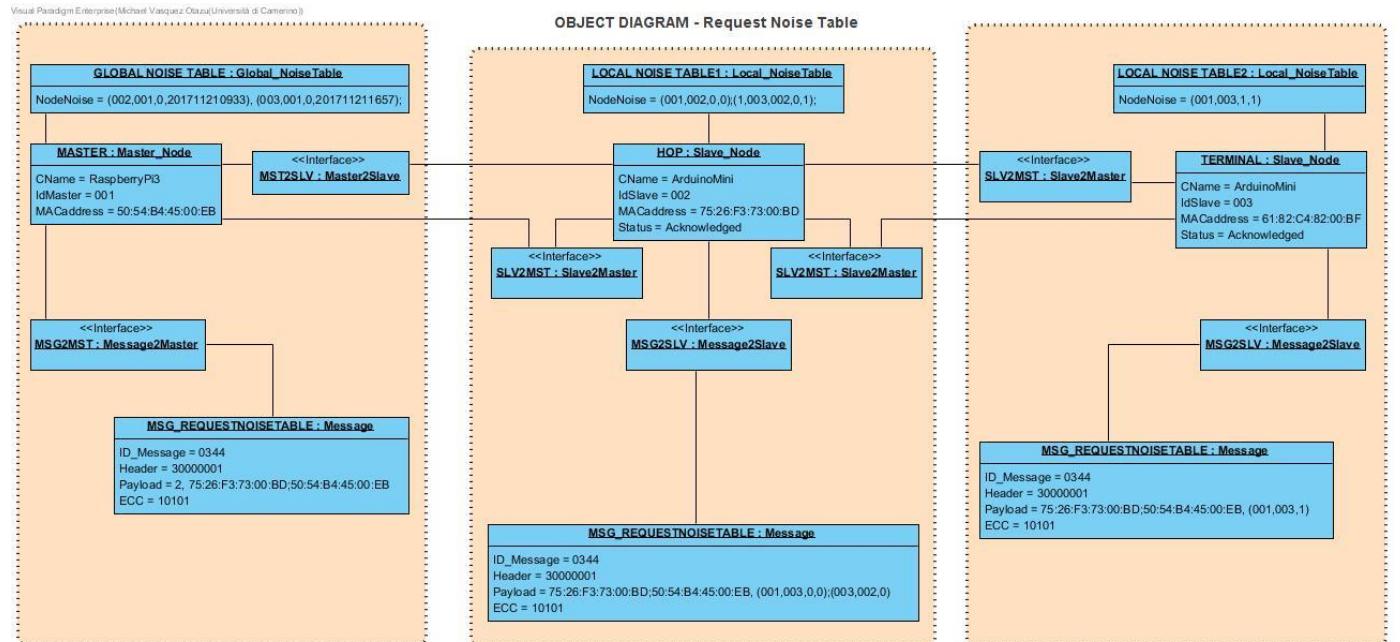
5.3 OBJECT DIAGRAM

Gli **Object Diagram** (OD) sono molto simili al **Class Diagram**, e come esso essi sono dei diagrammi di tipo **statico** che hanno come scopo descrivere il sistema in termini di **oggetti** e relative **relazioni** che sono **istanziate** in un determinato momento.

5.3.1 OD Hello



5.3.2 OD Request Noise Table



5.3.3 OD Send/Response

