POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Tecniche consistenti di whitelisting probabilistico in reti TSCH

Politecnico di Torino

Relatori

Dr. Stefano Scanzio

Dr. Gianluca Cena

Candidato

Federico Bitondo

Aprile 2022

Sommario

Lo standard Time Slotted Channel Hopping (TSCH) consente ai nodi in una rete mesh di comunicare in maniera robusta e deterministica, fornendo tecniche atte a mitigare efficacemente gli effetti delle interferenze esterne causate, in particolare, dal traffico generato da reti poste nelle vicinanze e a ridurre drasticamente i consumi energetici, garantendo la possibilità di costruire apposite reti di dispositivi autonomi e alimentati a batteria, dette Wireless Sensor Networks (WSN). L'introduzione della tecnica di channel hopping consente di raggiungere tali risultati: tentativi ripetuti di trasmissione del frame hanno luogo su canali logici, e quindi fisici, differenti. In questo modo, se l'implementazione assicura un'appropriata esplorazione dei canali a disposizione, l'effetto di un'interferenza esterna non si ripercuote sui tentativi di trasmissione successivi. Le tecniche di whitelisting possono essere sfruttate per escludere in modo dinamico i canali peggiori, riducendo le latenze e il numero medio di ritrasmissioni ed aumentando il livello di stabilità della rete. Per assicurare un'efficacia maggiore, le whitelist vengono definite in relazione ai singoli link, piuttosto che a livello globale, e sono associate alle specifiche coppie di nodi. L'informazione contenuta nelle whitelist deve essere scambiata in modo coerente in quanto entrambi i nodi devono concordare sul canale da utilizzare per la comunicazione. In caso contrario, i nodi potrebbero non essere in grado di comunicare e potrebbe verificarsi una disconnessione di una porzione della rete. In questo lavoro di tesi vengono valutati i requisiti di robustezza richiesti dal processo di scambio della whitelist e viene proposto uno specifico protocollo, denominato CONSIP, che fornisce una soluzione ai problemi citati in precedenza. Lo scambio di informazioni relative alla whitelist, infatti, avviene su due canali di comunicazioni distinti e paralleli, resi disponibili rispettivamente tramite l'utilizzo di celle principali e celle di backup che assicurano, al termine della transazione, che l'informazione scambiata sia la stessa sui due end-point. Allo stesso tempo, tuttavia, la ridondanza introdotta dall'utilizzo delle celle di backup provoca un leggero aumento nei consumi energetici misurati sul nodo ricevitore. L'altro contributo offerto da questa tesi è relativo all'implementazione di un algoritmo di whitelisting probabilistico che ha lo scopo di fornire gli strumenti per selezionare in maniera pseudo-randomica i canali su cui trasmettere. La selezione è basata su specifici parametri di qualità dei link misurati su apposite finestre temporali, di lunghezza configurabile. Il protocollo CONSIP

e il protocollo di whitelisting probabilistico sono stati verificati formalmente e per mezzo di apposite simulazioni, eseguite utilizzando una versione opportunamente modificata di un simulatore di reti TSCH implementato in Python. Un vasto numero di simulazioni ha permesso di sperimentare l'efficacia e le performance delle tecniche proposte. I parametri utilizzati per le misurazioni sono stati definiti in seguito a modellazioni realistiche dello stato dei canali oppure ricavati da un apposito database costruito a partire dall'analisi del traffico tra due nodi TSCH posizionati in un contesto reale di laboratorio e circondati da sorgenti di interferenze esterne. I risultati delle simulazioni hanno sottolineato un incremento nullo o trascurabile in termini di consumi energetici e latenze.

Indice

Introduzione 7

1 TSCH 10
1.1 Wireless Sensor Networks . . . 10
1.2 Standard . . . 13
1.2.1 Slotframe . . . 14
1.2.2 Channel Hopping . . . 17
1.2.3 Latenze e sincronizzazione . . . 20
1.3 Stack protocollare . . . 23
1.3.1 6TiSCH . . . 24
1.3.2 6LoWPAN . . . 25
1.3.3 RPL . . . 27
1.3.4 CoAP . . . 29
1.4 Blacklisting e Whitelisting . . . 31

2 Protocollo di consistenza 35
2.1 Introduzione . . . 35
2.2 CONSIP . . . 38

3 Probabilistic listing 45

4 Implementazione 51
4.1 OpenWSN . . . 51
4.1.1 Software di base . . . 51
4.1.2 Modifiche apportate . . . 53
4.2 Aggregazione e analisi dei dati raccolti . . . 59
4.3 TSCHpredictor . . . 60
4.3.1 Software di base . . . 60
4.3.2 Modifiche apportate . . . 62

5 Risultati 73
5.1 Setup e metriche di interesse . . . 73
5.2 Simulazioni protocollo CONSIP . . . 75

5.3 Simulazioni protocollo di whitelisting probabilistico 82

Conclusioni 87

Elenco delle figure 88

Elenco delle tabelle 89

Elenco dei listati 90

Glossario 91
Acronimi 91
Termini 93

Bibliografia 95

Introduzione

La sempre maggiore disponibilità di dispositivi a basso costo alimentati autonomamente a batteria e la necessità di comunicare in maniera capillare con tutti gli elementi di un sistema aziendale hanno favorito lo sviluppo delle reti Wireless Sensor Network (WSN), largamente utilizzate anche in contesti urbani, medici e militari. I nodi che formano la rete WSN hanno il compito di trasmettere i risultati delle misurazioni periodiche effettuate oppure di attuare delle azioni nei confronti del sistema impartite da uno o più nodi coordinatori, tipicamente dotati di maggiore potenza di calcolo e minori vincoli dal punto di vista energetico. I dispositivi vengono spesso posizionati in zone difficilmente accessibili e, quindi, è necessario estendere il più possibile l'indipendenza energetica dei nodi. A tale scopo è stato introdotto il protocollo Time Slotted Channel Hopping (TSCH), incluso nello standard IEEE 802.15.4 a partire dal 2012 e caratterizzato dall'utilizzo di una tecnica di accesso al canale basata sulla suddivisione del tempo in intervalli temporali di lunghezza limitata, detti *timeslot*, e sull'utilizzo della tecnica di *channel hopping*, orientata a sfruttare la diversità di frequenza per garantire probabilità di successo di trasmissione di frame successivi scorrelate tra loro. La discretizzazione del tempo e la possibilità di selezionare il canale logico di trasmissione si concretizzano nella definizione di celle di comunicazione che formano una struttura dati bidimensionale, detta *slotframe*. Le celle all'interno dello slotframe definiscono lo *schedule* di comunicazione tra i vari nodi e vengono ripetute periodicamente su slotframe successivi. Le celle vengono univocamente identificate per mezzo della coppia di indici *channel offset* e *slot offset*. Il protocollo TSCH consente di porre i nodi in uno stato di *sleep* nel caso in cui non siano coinvolti in alcuna operazione di trasmissione o ricezione secondo quanto espresso dallo schedule. Si è misurato sperimentalmente che i nodi in una rete TSCH si trovano in uno stato di inattività per oltre il 99.9% del loro tempo di vita. In questo modo è possibile estendere l'autonomia dei nodi a valori nell'ordine degli anni.

Le operazioni necessarie per la corretta comunicazione tra due nodi variano a seconda del ruolo assunto dai due dispositivi. Il nodo trasmettitore, infatti, ha la possibilità di concludere anticipatamente le *routine* di trasmissione in caso di assenza di pacchetti nella coda di trasmissione relativa al nodo destinatario. Quest'ultimo, non avendo visibilità dello stato della coda di trasmissione dell'altro nodo,

entra in uno stato di ricezione che viene ripetuto periodicamente in occorrenza della cella schedulata in questione, anche in assenza di pacchetti trasmessi. Tale fenomeno prende il nome di *idle listening* e provoca, in generale, un maggior consumo energetico misurato sui nodi che operano in modalità di ricezione.

Numerosi studi hanno dimostrato che l'utilizzo incondizionato di tutti i canali a disposizione nella porzione di banda stabilita dallo standard IEEE 802.15.4 porta a risultati non ottimi. Le reti WSN, infatti, operano in un contesto esposto ad interferenze esterne di tipo *wireless* provocate da sorgenti di varia natura. Dato che le interferenze tipicamente coinvolgono un sottoinsieme dei canali a disposizione, sono state proposte nel tempo diverse tecniche di *whitelisting* o *blacklisting*, orientate rispettivamente ad includere i canali che hanno dimostrato una maggiore affidabilità o escludere i peggiori.

In letteratura, tuttavia, si è rivolta spesso una maggiore attenzione ai criteri di implementazione di tali tecniche rispetto alle metodologie di trasmissione coerente dell'informazione. In questo lavoro di tesi si affronta suddetto tema attraverso l'analisi dei requisiti necessari a garantire la coerenza della whitelist ai due estremi della comunicazione e viene proposto un algoritmo, denominato CONSIP, che fornisce un modello di scambio coerente della configurazione di rete aggiornata. Questa tecnica prevede l'introduzione di un'ulteriore cella di supporto per ogni cella schedulata nello slotframe, da attivare esclusivamente durante le fasi di scambio della whitelist. CONSIP fornisce esclusivamente gli strumenti per instaurare un canale di comunicazione temporaneo robusto ed è, dunque, trasparente rispetto alla specifica implementazione della whitelist o blacklist trasmessa.

Per questo motivo si è deciso di proporre, in aggiunta, un algoritmo di whitelisting probabilistico che, a differenza dei classici algoritmi basati sulla totale inclusione o esclusione di un canale dalla lista di quelli utilizzabili, introduce una priorità relativa di estrazione del canale da utilizzare per la trasmissione. Il vantaggio diretto che ne deriva è relativo alla possibilità di adattare in maniera dinamica la whitelist alle condizioni istantanee della rete, associando una maggiore probabilità ai canali che hanno performato meglio e, allo stesso tempo, lasciando una ridotta probabilità di estrazione a quelli peggiori, che potrebbero ritornare ad un livello di qualità maggiore nelle finestre temporali successive. In altre parole, il protocollo consente di reintrodurre automaticamente, e secondo parametri configurabili, i canali in precedenza penalizzati evitando di introdurre delle euristiche relative alle politiche di reinserimento all'interno della whitelist.

Lo studio teorico dei protocolli è stato affiancato da due fasi pratiche: il monitoraggio di uno scenario di rete reale con successiva creazione di database contenenti le informazioni raccolte, eseguito tramite la modifica e il caricamento del codice sorgente del progetto OpenWSN su board a disposizione nell'ambiente di laboratorio, e l'estensione di un simulatore di reti TSCH, denominato `TSCHpredictor` e sviluppato in Python, in modo da implementare e valutare sperimentalmente i protocolli proposti.

La tesi presenta la seguente struttura. Nel Capitolo 1 vengono introdotte in maggiore dettaglio le reti WSN, lo standard TSCH e lo stack protocollare che consente di comunicare con i dispositivi attraverso la rete IPv6. Il Capitolo 2 e Capitolo 3 contengono l'analisi dei requisiti e la descrizione formale rispettivamente del protocollo CONSIP e del protocollo di whitelisting probabilisico. La descrizione degli strumenti utilizzati e delle modifiche apportate agli stessi è contenuta nel Capitolo 4, mentre i risultati delle campagne di simulazione sono presentati e discussi in Capitolo 5. Nella sezione Conclusioni, per ultimo, vengono formulate delle considerazioni finali e vengono presentate alcune possibilità di sviluppi futuri.

Capitolo 1

TSCH

1.1 Wireless Sensor Networks

Nell'ultimo decennio si è assistito ad un incremento esponenziale della mole di dati generata ed utilizzata da sistemi di varia natura. Un noto articolo sull'argomento [1] identifica tre dimensioni di crescita dei dati, le cosiddette $3V$: volume, velocità e varietà. La mole di dati tende a crescere a ritmo elevato, da qui la necessità di sistemi in grado di immagazzinare tali informazioni. La velocità fa riferimento sia alla frequenza della disponibilità di nuovi dati sia alla rapidità di trasmissione degli stessi. Si prenda come esempio un sistema real-time critico che, per design, debba garantire tempi di reazione limitati in risposta a stimoli provenienti dall'esterno: in questo scenario appare evidente come la velocità di trasmissione e la robustezza della comunicazione siano di primaria importanza. Infine, le sorgenti di dato e l'informazione stessa possono essere di varia natura.

In ambito industriale numerosi sforzi e finanziamenti sono stati dedicati allo studio di sistemi, interconnessi in modalità *wireless* [2] e non necessariamente di piccolo raggio, in grado di osservare il fenomeno di interesse e convogliare le informazioni verso un unico punto di controllo che, all'occorrenza, può aggregarle e restituire un *feedback* al resto del sistema sotto forma di operazioni da attuare. Quando il basso consumo energetico è essenziale e la quantità di dati è ridotta una tecnologia molto utilizzata è la Wireless Sensor Network (WSN), o più generalmente Wireless Sensor and Attuator Network (WSAN), nel caso in cui i dispositivi siano anche in grado di attuare delle azioni nei confronti del sistema.

Le reti WSN sono formate da una serie di sensori, detti anche *nodi* o *mote*, che comunicano tra di loro e con il coordinatore attraverso l'etere. Ciò garantisce una maggiore flessibilità del sistema, dato che i sensori possono essere posti in punti altrimenti inaccessibili in caso di collegamento via cavo. Come diretta conseguenza i sensori, dopo una fase di inizializzazione, lavorano in autonomia e sono alimentati a batteria.

Tali reti trovano applicazione nei più svariati campi, tra cui il monitoraggio ambientale, l'agricoltura di precisione, l'urbanistica, la sorveglianza e l'ambito militare [3, 4, 5]. I sensori maggiormente utilizzati possono essere raggruppati in cinque categorie principali: ambientali (temperatura, umidità, vento), gas (inquinamento, CO_2), fisici (accelerometro, ultrasuoni, vibrazione), ottici (infrarossi, ultravioletti, raggi solari) e biometrici (battito cardiaco, pulsossimetria, sudore).

Tra le applicazioni ambientali più interessanti troviamo il monitoraggio dell'habitat naturale di particolari specie animali. I ricercatori, infatti, stanno acquisendo sempre più consapevolezza sull'impatto della presenza umana sull'ambiente durante le operazioni di monitoraggio e rilievo. A titolo di esempio, le attività di ricerca nelle isole Great Duck Island, nel golfo del Maine, hanno suggerito che le colonie di uccelli marini che popolano la zona durante i periodi caldi dell'anno soffrono notevolmente della presenza umana, a tal punto che anche una breve visita, nell'ordine dei minuti, può provocare un tasso di mortalità del 20% tra uova e pulcini [6]. L'utilizzo di reti di sensori ha permesso l'acquisizione di statistiche rilevanti, quali la temperatura delle tane di nidificazione o la rilevazione di movimento, preservando la quiete dell'ambiente e ottenendo, allo stesso tempo, un risparmio in termini economici dovuto alla strategia *"deploy 'em and leave 'em"*, per cui gli sforzi logistici sono limitati all'installazione iniziale e ad una manutenzione occasionale. Altri esempi di questo tipo sono il tracciamento delle abitudini e degli spostamenti delle zebre in Kenya (ZebraNet [7]) e lo studio di fenomeni naturali in ambienti ostici, quali l'analisi del movimento dei ghiacciai ([8]), il monitoraggio dello stato di vulcani attivi ([9]) e la misurazione dei livelli di radioattività nell'aria ([10]).

In ambito medico, queste tecnologie hanno consentito di sperimentare tecniche di monitoraggio in tempo reale dei parametri vitali di un paziente e di segnalazione immediata ai centri di soccorso in caso di rilevazione di incidenti. In particolare, per ottenere tali misurazioni, ai pazienti vengono forniti appositi dispositivi indossabili, possibilmente non intrusivi, che formano le cosiddette Body Sensor Network (BSN).

A conclusione degli esempi di applicazione delle reti WSN si possono citare gli utilizzi in ambito militare e urbanistico. Nel primo caso i dispositivi vengono utilizzati sia per il monitoraggio (delle truppe alleate, delle condizioni di una zona militare, della presenza di munizioni e rifornimenti) sia per il tracciamento e la sorveglianza (delle truppe nemiche, del passaggio di mezzi militari, della presenza di fenomeni acustici riconducibili ad azioni militari) [11]. Nel secondo caso le applicazioni sono innumerevoli e prevalentemente orientate alla costruzione di *smart cities* attraverso operazioni, ad esempio, di controllo della congestione del traffico, della presenza di parcheggi liberi, dello stato strutturale di edifici, del livello di rifiuti e di inquinamento acustico e ambientale [12].

I tre indici prestazionali principali di un sistema WSN sono l'affidabilità, la latenza ed il consumo energetico [13, 14]. Affidabilità e latenza sono due metriche che descrivono la robustezza di un sistema. La prima, requisito presente nella maggior parte dei sistemi, definisce il grado di tolleranza alla perdita di pacchetti scambiati

tra due nodi. L'intera rete è posta in un'ecosistema aperto, ovvero esposto – anche involontariamente – a disturbi ed interferenze esterne. In particolare, nelle reti in questione la comunicazione avviene via etere e, quindi, il fenomeno di perdita o corruzione irreversibile dell'informazione è assolutamente non trascurabile e causato principalmente dai fenomeni di interferenza di tipo *narrow-band* (sovrapposizione con reti IEEE 802.11 vicine) e di *multipath fading* (più repliche di un segnale, a diversa fase, che giungono alla stessa destinazione causando l'attenuazione o la distorsione del segnale stesso) [15]. Una rete WSN robusta dev'essere in grado di rimediare all'instabilità della comunicazione introducendo, ad esempio, tecniche di ritrasmissione dell'informazione persa. In determinate applicazioni denominate *real-time*, inoltre, non è sufficiente garantire la ricezione di un'informazione, ma è necessario farlo entro intervalli temporali ben definiti. Ciò viene formalizzato esprimendo dei requisiti in termini di latenza. Va ricordato che le reti WSN sono spesso utilizzate per operazioni di monitoraggio ambientale a cui possono seguire, a seconda dei casi, segnalazioni ad altri sistemi predisposti o attuazioni dirette di determinate azioni in risposta, ad esempio, a misure non coerenti con i valori normali. La prontezza dello scambio dell'informazione garantisce la realizzabilità delle operazioni di controllo e attuazione.

L'ultimo dei tre parametri è l'indipendenza energetica dei nodi. Generalmente i sensori consistono in microcontrollori dotati di risorse limitate e antenne conformi allo standard IEEE 802.15.4 per la comunicazione. Sono inoltre alimentati a batteria, come già detto, per cui le componenti hardware installate e il software eseguito devono limitare al massimo i consumi. Per questo motivo i nodi che svolgono le operazioni di *sensing*, comunemente detti nodi foglia o *leaf node*, si limitano periodicamente ad accendere le componenti necessarie per la rilevazione e a comunicare il risultato ad altri attori della rete, tipicamente calcolatori con maggiore potenza di calcolo e minori requisiti sul piano dell'efficienza energetica, in quanto spesso collegati costantemente a fonti di alimentazione. In media, i nodi foglia trascorrono più del 99% del tempo in stato di *sleep* [16] e, nella maggior parte degli scenari, raccolgono e trasmettono piccole quantità di informazione (nell'ordine della decina di byte) con una periodicità costante (nell'ordine delle decine di secondi).

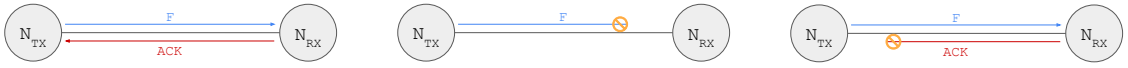
1.2 Standard

Il protocollo Time Slotted Channel Hopping (TSCH) è stato proposto come *amendment* allo standard IEEE 802.15.4 (IEEE 802.15.4e) nel 2012 ed introdotto definitivamente nelle versioni del 2015 e del 2020 [17]. Definisce una modalità operativa di accesso al canale basata sull'accesso multiplo a divisione di tempo denominata Time Division Multiple Access (TDMA) e sulla diversità di frequenza, garantendo alle applicazioni di rete in ambito industriale elevate affidabilità e ridotti consumi energetici, inquadrandosi perfettamente nel contesto delle reti Low-Rate Wireless Personal Area Networks (LR-WPAN).

A differenza delle reti Wi-Fi, nelle quali i dispositivi dispongono di una banda più ampia e di una maggiore potenza trasmissiva, lo standard TSCH sfrutta al massimo le limitate risorse dei dispositivi che lo implementano limitando la dimensione dei *frame* scambiati (127 B, dei quali circa 100 utilizzabili dai livelli superiori nello stack protocollare) e il rate di trasmissione (derivabile dalla durata del singolo timeslot, come in Sezione 1.2.1).

Nelle reti TSCH ogni frame trasmesso viene confermato a livello MAC da un messaggio di ACK di lunghezza fissa (Figura 1.1). Il nodo ricevitore invia il frame di ACK (a) solamente nel caso in cui sia stato in grado di ricevere ed interpretare correttamente il frame di dato (ad esempio attraverso degli algoritmi di rilevazione dell'errore sul bit). Nel caso opposto (b) non trasmette alcun messaggio al nodo trasmettitore, che prende atto della mancata conclusione dell'intera transazione e, per un numero limitato di volte, ritenta la trasmissione. La stessa cosa avviene in caso di perdita del messaggio di ACK stesso (c). Il nodo trasmettitore (che nella seconda fase della transazione assume il ruolo di ricevitore dell'acknowledge), in definitiva, non è in grado di determinare la causa del mancato arrivo dell'ACK.

Questo è un problema tipico delle reti, soprattutto quelle wireless che presentano un'alta probabilità di errore. In alcune implementazioni si utilizza la tecnica opposta, ovvero si notifica esplicitamente la perdita di un frame attraverso dei messaggi di Negative ACK (NACK). L'assenza di questi ultimi implica la corretta ricezione del frame.



(a) data ACK (b) no data (c) data no ACK

Figura 1.1: Trasmissione frame ed eventuale ACK

1.2.1 Slotframe

Lo standard introduce una divisione del tempo in intervalli di eguale durata, detti *timeslot*, nei quali vengono effettuate tutte le operazioni relative alla singola trasmissione o ricezione di un frame di dato. La durata del timeslot varia a seconda delle implementazioni, con valori tipici nell'ordine della decina di millisecondi. I dispositivi che implementano TSCH devono obbligatoriamente concludere l'intero ciclo di operazioni, cioè la trasmissione del dato e del relativo ACK, entro questo intervallo di tempo.

Lo *slotframe*, invece, è una struttura multidimensionale che comprende un determinato numero di timeslot ripetuti nel tempo e caratterizzati dalla possibilità di utilizzare uno specifico numero di canali. Ne deriva una matrice di dimensione $N_{slots} \times N_{ch}$, con N_{slots} pari al numero di timeslot e N_{ch} pari alla cardinalità del sottoinsieme di frequenze, fisiche o logiche, che possono essere utilizzate simultaneamente per la comunicazione. A titolo di esempio, il progetto open-source OpenWSN – che implementa un intero stack protocollare (Sezione 1.3) in cima allo standard TSCH – utilizza un numero di slot $N_{slots} = 101$, $N_{ch} = 16$ canali logici differenti e una durata del singolo timeslot pari a 20 ms [18]. In questo caso la periodicità dell'intero slotframe è data da $N_{slots} \times 20\,\text{ms} = 2.02\,\text{s}$.

La matrice, che può essere più o meno sparsa, rappresenta lo *schedule* di comunicazione della rete, o di una porzione della stessa. Ogni cella schedulata all'interno dello slotframe identifica una comunicazione unidirezionale tra una coppia di nodi, caratterizzata da un nodo trasmettitore N_{TX} e un nodo ricevitore N_{RX}. I nodi si portano in uno stato attivo, che sia di ricezione o di trasmissione, solamente in corrispondenza di una cella schedulata che li coinvolge. In caso contrario protraggono il proprio stato di sleep fino alla successiva comunicazione schedulata. È fondamentale che i nodi siano sincronizzati con sufficiente precisione fra di loro, in modo da garantire che il nodo N_{RX} si metta in ascolto nel momento in cui N_{TX} trasmette un frame (Sezione 1.2.3). È possibile schedulare diverse trasmissioni sia lungo la dimensione orizzontale della matrice, ponendole dunque in momenti temporali ben distinti, sia lungo la dimensione verticale, ammettendo una sovrapposizione di trasmissioni su frequenze logiche – e quindi fisiche – diverse all'interno dello stesso timeslot. Sfruttando la seconda proprietà si riescono a garantire fino a N_{ch} trasmissioni simultanee tra altrettante coppie di nodi N^i_{TX} e $N^i_{RX} \in \mathcal{T}$ (insieme dei nodi coinvolti in tutte le trasmissioni nel timeslot fissato), con $i \in \{0, ..., N_{ch} - 1\}$ rappresentante la frequenza logica della comunicazione e N^i_{TX} e N^i_{RX} rispettivamente il nodo trasmettitore e ricevitore della i-esima trasmissione, a condizione che ogni nodo sia coinvolto una sola volta nelle comunicazioni, a prescindere dal proprio ruolo. In altre parole, più coppie di nodi sono in grado di comunicare contemporaneamente e senza interferenze sfruttando range di frequenze differenti all'interno della banda disponibile, con l'unica e ovvia limitazione di non coinvolgere lo stesso nodo in più trasmissioni simultanee.

In questo modo si riescono ad ottenere schemi di comunicazione molto fitti che possono essere sfruttati, ad esempio, in sistemi industriali ad elevata densità di dispositivi. È possibile, in aggiunta, configurare alcune celle in modalità *shared* ed utilizzarle per scopi di configurazione e controllo della rete (Sezione 1.3). Esse operano in modalità *slotted aloha*, per cui non presentano le peculiarità tipiche delle celle TX e RX.

La cella è identificata dalla coppia di valori *slot offset (s)* e *channel offset (c)* e si ripete periodicamente ogni N_{slots} timeslot. In presenza di vincoli più stringenti in termini di latenza e/o robustezza è possibile schedulare più celle, all'interno dello slotframe, per la comunicazione tra la stessa coppia di nodi. La tecnica dell'*overprovisioning*, ad esempio, prevede l'allocazione di un numero maggiore di celle rispetto al necessario allo scopo di limitare le latenze in caso di perdita – con successiva ritrasmissione – di frame e di garantire una soglia minima di *throughput* in caso di repentino aumento del numero di pacchetti nelle code. In ogni caso, si viene a formare un insieme di celle, detto *bundle*, che può avere dimensione fissa o, grazie all'utilizzo di algoritmi specifici, variabile in base alle necessità istantanee della rete. In caso di sovrallocazione di celle l'overprovisioning può portare ad un aumento notevole dei consumi. Gli algoritmi adattivi hanno, dunque, il compito di aggiungere o rimuovere delle celle tenendo traccia dell'effettivo utilizzo delle stesse [19] o, in alcuni casi, disabilitarle in caso di assenza di comunicazione precedentemente concordata tra i nodi [20].

In seguito alle fasi di *bootstrap* del sistema, il tempo viene discretizzato e si assegna un identificatore univoco e monotonicamente crescente allo slot di riferimento. Tale contatore, che prende il nome di Absolute Slot Number (ASN), viene costantemente aggiornato dai dispositivi e memorizzato su un numero di bit sufficiente a garantire il non *overflow* per un intervallo di tempo sufficientemente lungo. In OpenWSN, ad esempio, l'ASN è memorizzato su 5 B, estendendo fino a circa 700 anni la durata di vita del sistema.

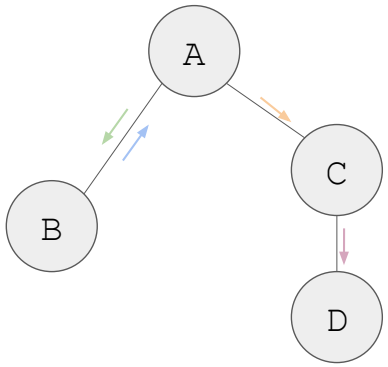


Figura 1.2: Esempio di topologia di rete

Prendiamo ora in considerazione uno scenario semplice, composto da quattro nodi disposti secondo la topologia raffigurata in Figura 1.2. Il nodo A è in grado di comunicare direttamente con il nodo B e il nodo C che, a sua volta, ha la possibilità di trasmettere (ricevere) frame al (dal) nodo D. A partire dalla topologia di rete è possibile generare uno schedule che soddisfi le esigenze dell'applicazione. Ad esempio, in Figura 1.3 è riportato uno slotframe nel quale sono state schedulate 4 celle per la comunicazione tra i nodi. Le dimensioni della matrice sono $N_{slots} = 4$ e $N_{ch} = 3$. Si noti la replica nel tempo di tali slotframe, come da standard, con i singoli timeslot identificati univocamente e globalmente per mezzo dell'ASN. I dati nella rete possono seguire due percorsi: $(A \rightarrow B)$ e $(B \rightarrow A \rightarrow C \rightarrow D)$, anche raffigurati – per mezzo di colori differenti – in Figura 1.2. È impossibile, ad esempio, trasmettere un frame dal nodo D al nodo C.

A completamento dell'esempio, si supponga di voler trasmettere un frame di dato dal nodo B al nodo D, generato in corrispondenza dell'ASN 35 (non in figura) e, quindi, già presente nella coda del nodo B al tempo 36. La prima trasmissione avviene, dal nodo B al nodo A, in corrispondenza della cella $(1, 1)$ (notazione per rappresentare gli indici (s, c)) e $ASN = 37$. Il nodo A può inoltrare il frame nella cella $(3, 2)$, $ASN = 39$ e, infine, il nodo C può completare la trasmissione recapitando il pacchetto al nodo D in $(1, 2)$, $ASN = 41$. Le altre celle non vengono utilizzate per trasmettere informazione.

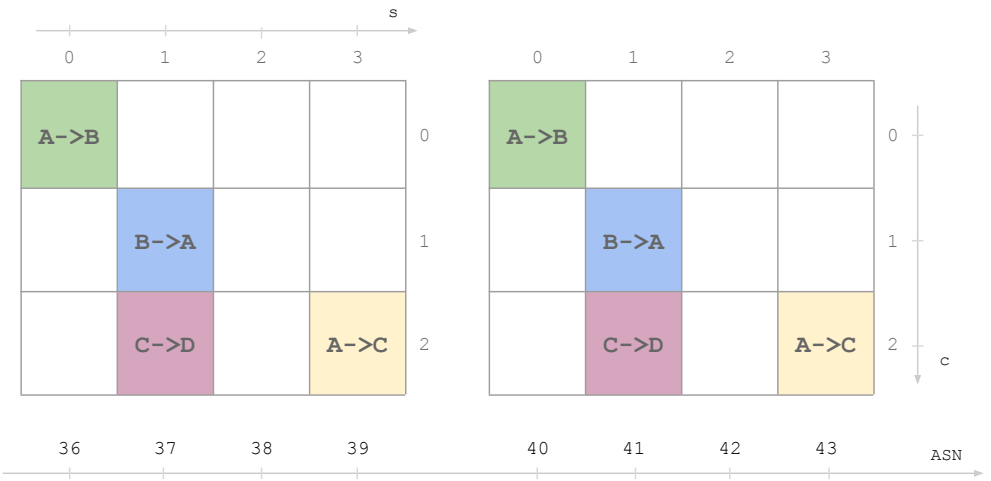


Figura 1.3: Esempio di slotframe con visibilità globale. Ogni nodo ne memorizza una versione con le sole celle che lo riguardano.

È bene sottolineare che ogni nodo mantiene la propria versione locale dello slotframe. Il nodo B in Figura 1.3, ad esempio, memorizza una matrice con le sole celle $(0, 0)$ e $(1, 1)$, con abbinate le operazioni ad esse associate (rispettivamente ricezione e trasmissione). Ciò comporta diverse implicazioni e criticità. La prima

osservazione è legata al ruolo ricoperto dai due nodi posti all'estremo della comunicazione. La cella ($N_{TX} \rightarrow N_{RX}$), ad esempio, ha l'effetto di preparare il nodo N_{TX} per la trasmissione e, in modo complementare, il nodo N_{RX} per la ricezione. Questo richiede l'esecuzione di routine diverse sui due mote. In particolare, sul nodo N_{TX} vengono effettuate delle operazioni preliminari di risveglio dallo stato di sleep e, se necessario, la circuiteria atta alla trasmissione viene opportunamente alimentata e configurata e la trasmissione ha inizio. In caso di code vuote il nodo può decidere di ritornare prematuramente nello stato di sleep assicurando un risparmio in termini energetici. All'altro estremo della comunicazione, invece, il nodo N_{RX} si pone sempre in ascolto sulla frequenza stabilita, a prescindere dal comportamento di N_{TX}. Nel caso di effettiva trasmissione, il nodo ricevitore – a fronte di un certo dispendio di energia – può memorizzare e interpretare l'informazione recepita. In caso opposto, il nodo entra in uno stato di *idle listening* durante il quale – pur consumando energia – non viene ricevuta alcuna informazione. In alcune ottimizzazioni (si veda la Sezione 1.2.3) N_{RX} è in grado di rilevare l'assenza di trasmissione e riportarsi in uno stato di sleep con qualche millisecondo di anticipo, con conseguente – leggero – risparmio energetico.

Lo standard non specifica le modalità di costruzione e distribuzione dello slotframe, lasciando aperte diverse possibilità agli implementatori. Sebbene sia possibile costruire una versione globale dello schedule affidandosi ad un coordinatore (tipicamente il nodo root) che configura e distribuisce in maniera *top-down* lo slotframe a tutti i nodi della rete, la soluzione più efficiente e flessibile delega la costruzione dello slotframe ai nodi che desiderano instaurare nuove connessioni. La versione aggiornata della matrice sarà poi distribuita ai dispositivi vicini, che propagheranno le modifiche applicate allo schedule al resto della rete. Uno dei vantaggi della costruzione *bottom-up* dello slotframe è la possibilità di creare – in maniera più flessibile rispetto all'approccio duale – diverse sottoreti, collegate tra loro attraverso nodi di frontiera e dislocate spazialmente, che contribuiscono a mantenere lo schedule locale meno denso ed isolare logicamente zone distinte del sistema. Lo scopo di tutti gli algoritmi di scambio dello slotframe è quello di garantire la coerenza delle informazioni sullo schedule tra i nodi in ogni istante, in modo tale da evitare disconnessioni, locali o globali, all'interno della rete.

1.2.2 Channel Hopping

Una caratteristica di TSCH è la cosiddetta tecnica di *channel hopping* che, basandosi sul concetto di *channel diversity*, consente di utilizzare canali logici diversi per trasmissioni consecutive tra una coppia di nodi. Questo consente di mitigare l'effetto degli errori di trasmissione, causati da interferenze di vario tipo, utilizzando, secondo uno schema definito, un pattern di frequenze che garantisce di non soffrire a breve termine dell'effetto della stessa interferenza. Tipicamente i *link* di una rete sono coerenti sull'asse temporale. Questo significa che si può supporre

una certa regolarità nella probabilità di errore su uno stesso canale: in altre parole, la perdita di un frame su un canale ha un'alta probabilità di riverificarsi nelle ritrasmissioni successive. Allo stesso tempo i link sono indipendenti tra loro: se un frame è stato perso su uno specifico canale, la sua ritrasmissione su un'altra frequenza ha una probabilità di successo che non dipende dai precedenti tentativi, a patto che la sorgente dell'interferenza non sia in grado di influenzare più canali contemporaneamente.

TSCH garantisce la diversità di frequenza attraverso la tecnica del *channel hopping*: la cella $(N_{TX} \rightarrow N_{RX})$ – identificata per mezzo della coppia $(s,\ c)$ e dell'ASN di riferimento – trasmetterà su frequenze logiche sempre differenti, ricavate attraverso la seguente formula:

$$ch = H[(ASN + c)\ mod\ |H|] \tag{1.1}$$

con H che rappresenta la *hopping sequence*, ossia una permutazione degli $|H|$ canali logici definiti per la rete di riferimento. Tipicamente i valori di $|H|$ e N_{ch} coincidono (ossia, si possono utilizzare contemporaneamente tutti i canali a disposizione). Il canale ch è poi convertito, attraverso un mapping 1 a 1, nella frequenza fisica sulla quale le antenne vengono configurate. Tipicamente $N_{ch} = 16$ e le frequenze fisiche corrispondono alle sottobande definite dall'IEEE 802.15.4 a 2.4 GHz.

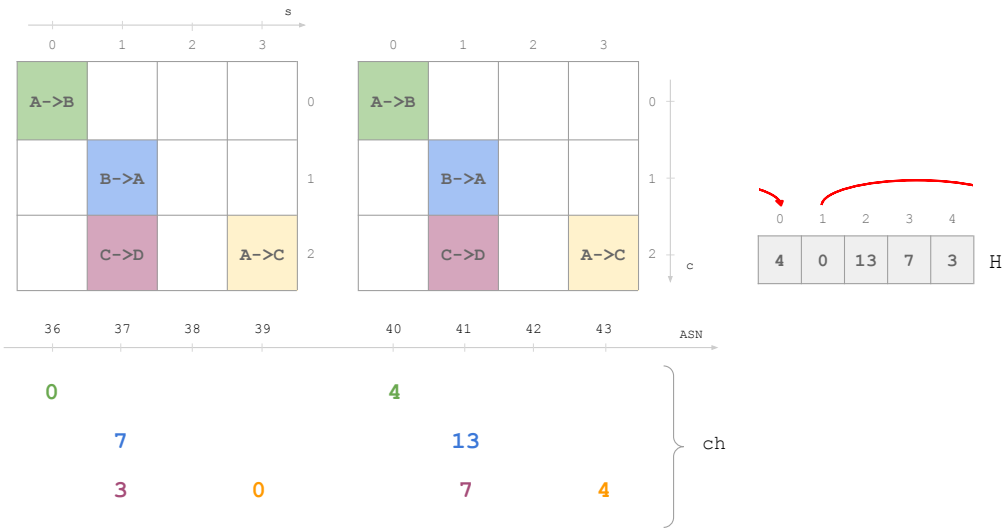


Figura 1.4: Channel hopping relativo alla sequenza H. Sono riportati i canali logici corrispondenti alle trasmissioni schedulate nello slotframe e ricavati per mezzo di Equazione (1.1). Con questi parametri, due trasmissioni successive nella stessa cella causano un salto nel vettore di frequenze pari a 4 celle (come indicato dalla freccia).

L'operatore di modulo in Equazione (1.1) garantisce il salto tra frequenze distinte a parità di *slot offset* e *channel offset*, ovvero su trasmissioni consecutive di frame in corrispondenza della stessa cella schedulata nello slotframe. Affinché ciò avvenga è necessario che $|H|$ e N_{slots} siano primi tra loro. In Figura 1.4 viene riportato un esempio di utilizzo della funzione di scelta del canale, basato sullo slotframe di Figura 1.3 e caratterizzato dai seguenti parametri: $N_{ch} = 3$, $N_{slots} = 4$, $H = [4, 0, 13, 7, 3]$, $|H| = 5$. In fase di implementazione, inoltre, è necessario definire la regola di conversione da ch a f. In questo caso, ad esempio, si può pensare di applicare un offset pari a 11, in modo tale da rappresentare con $ch = \{0, ..., 15\}$ le bande di frequenza $f = \{11, ..., 26\}$ definite in IEEE 802.15.4.

Per garantire la simultaneità delle trasmissioni in uno stesso timeslot è fondamentale che il vettore H sia lo stesso in tutti i nodi. Se ciò non fosse, la *hopping function* potrebbe estrarre la stessa frequenza per la trasmissione tra più coppie di nodi, provocando interferenze tra gli stessi e, in definitiva, aumentare la probabilità di disconnessione di una porzione della rete. Una scelta coerente con l'obiettivo di rendere indipendenti le probabilità di errore su ritrasmissioni successive è quella di garantire che queste avvengano su porzioni sufficientemente distanziate dello spettro disponibile. Tipicamente le interferenze sono dovute alla coesistenza delle reti TSCH con reti Wi-Fi, nelle quali i canali di trasmissione sono composti da un aggregazione di più sottobande della rete 2.4 GHz (Figura 1.5) e, quindi, spostarsi su un altro estremo dello spettro garantisce di non soffrire della stessa interferenza.

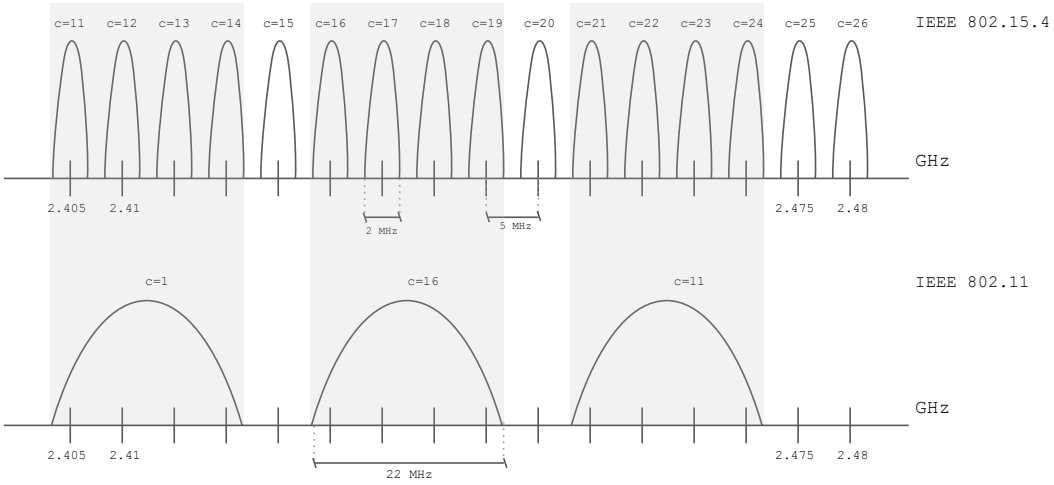


Figura 1.5: Frequenze nella banda 2.4 GHz in reti IEEE 802.15.4 e IEEE 802.11.

Come visto anche per la definizione dello slotframe, lo standard si limita ad introdurre la tecnica basata sul vettore di canali H, lasciando libertà di implementazione ai produttori. In OpenWSN $H = [5, 6, 12, 7, 15, 4, 14, 11, 8, 0, 1, 2, 13, 3, 9, 10]$

includendo, dunque, tutte le frequenze logiche ($N_{ch} = 16$). La traduzione in frequenza fisica è data semplicemente da $f_i = H[i] + 11 \quad \forall i \in \{0, ..., 15\}$.

L'effetto della *channel diversity* applicata a tutte le frequenze disponibili è quello di distribuire l'utilizzo nel tempo delle stesse ottenendo, di fatto, un valore medio di probabilità di errore. Se da un lato, infatti, l'utilizzo di una nuova frequenza può mitigare l'effetto di una precedente interferenza, è possibile andare incontro allo scenario opposto: il passaggio da una frequenza che ha garantito la corretta trasmissione (con conferma) del frame ad una con più alta probabilità di errore. Indicando con $\mathcal{C} = \{ch_1, ..., ch_n\}$ l'insieme di tutti i canali logici disponibili, la *fairness* introdotta dall'utilizzo di $H = perm(\mathcal{C})$[1] non porta all'ottimo in termini di Packet Delivery Ratio (PDR), metrica che rappresenta la percentuale di pacchetti correttamente trasmessi e confermati. In Sezione 1.4 si discutono tecniche esistenti di *whitelisting* e *blacklisting* che riducono la cardinalità di H, includendo solamente le migliori – secondo criteri da definire – frequenze disponibili o escludendo le peggiori. Nel Capitolo 3, invece, verrà proposta una tecnica alternativa finalizzata ad associare una probabilità di estrazione maggiore per le frequenze che dimostrano un valore di PDR migliore nella finestra temporale precedente.

1.2.3 Latenze e sincronizzazione

Il protocollo TSCH consente di ottenere un'elevata affidabilità (superiore al 99%, anche in ambienti fortemente soggetti ad interferenze esterne) e allo stesso tempo un basso consumo energetico. Questo perché i nodi trasmettono, come detto, su frequenze sempre differenti e per la maggior parte del loro tempo si trovano in stato di sleep, con la maggior parte dei circuiti hardware non alimentati. Restano attivi dei timer che consentono di rilevare l'inizio di una cella schedulata per il nodo e configurare il ricetrasmettitore per la trasmissione o ricezione del frame. Per poter comunicare in modo corretto è necessario che i nodi N_{TX} e N_{RX} entrino nei rispettivi stati operativi negli stessi istanti. In termini assoluti, è impossibile – anche disponendo di hardware di ottima fattura – garantire una precisione costante nell'ordine dei μs per un tempo indefinito. Ciò che si verifica nella realtà è una desincronizzazione relativa tra i nodi, che risulta in uno sfasamento degli assi temporali con conseguente impossibilità di comunicare. Tale fenomeno, detto *drift* dei timer interni, è accentuato dalla relativamente bassa qualità costruttiva dei dispositivi impiegati nelle reti TSCH, in particolare gli oscillatori al quarzo, e può dipendere da fattori interni (qualità delle componenti, errori in fase di produzione) e esterni (temperatura, voltaggi troppo bassi o elevati, invecchiamento dei device) [21]. Per rimediare al fenomeno intrinseco del drift dei timer, lo standard definisce

[1] Specifica permutazione degli elementi del vettore di frequenze, che garantisce la diversità spaziale

un protocollo di sincronizzazione relativa tra i nodi basato sullo sfruttamento del traffico dati: i nodi prendono nota del tempo effettivo di ricezione del primo bit di un frame e lo confrontano con il tempo di arrivo stimato, correggendo il proprio timer conseguentemente. Esistono due modalità di sincronizzazione: *packet-based* e *ACK-based* [17, 22]. Nella prima il nodo N_{RX}, riconoscendo una differenza tra il tempo stimato t_{RX}^{est} e il tempo effettivo t_{RX} di ricezione del primo bit, applica una correzione ai propri timer pari a $\pm t_{corr}$, con $t_{corr} = |t_{RX} - t_{RX}^{est}|$. All'atto pratico, la durata del timeslot viene ridotta in caso di $t_{RX} > t_{RX}^{est}$ (ossia, il nodo aveva abilitato con troppo anticipo la propria antenna per la ricezione) ed estesa in caso contrario (primo bit di informazione ricevuto prima del previsto). Nella seconda opzione, invece, il nodo N_{RX} – che funge da *time source* per N_{TX} – calcola il drift relativo nella maniera vista in precedenza ed include direttamente nel frame di ACK la correzione da applicare al nodo che ne ha fatto richiesta (implicita).

Più il traffico dati tra i nodi è elevato, più frequenti sono le correzioni ai timer e, dunque, maggiore la stabilità del sistema. Per ovviare ad una eventuale (anche solo limitata) mancanza di comunicazione dovuta all'assenza di pacchetti nelle code, i nodi possono trasmettere dei frame Keep Alive (KA) di sollecitazione della sincronizzazione oppure utilizzare meccanismi di sincronizzazione più avanzati e precisi che permettono di compensare anche la frequenza di oscillazione dei quarzi, rendendo la rete più robusta alla mancanza di frame di sincronizzazione per tempi relativamente lunghi [23, 24, 25].

In Figura 1.6 vengono mostrate nel dettaglio le operazioni che un nodo, trasmettitore o ricevitore, effettua nell'arco di un timeslot.

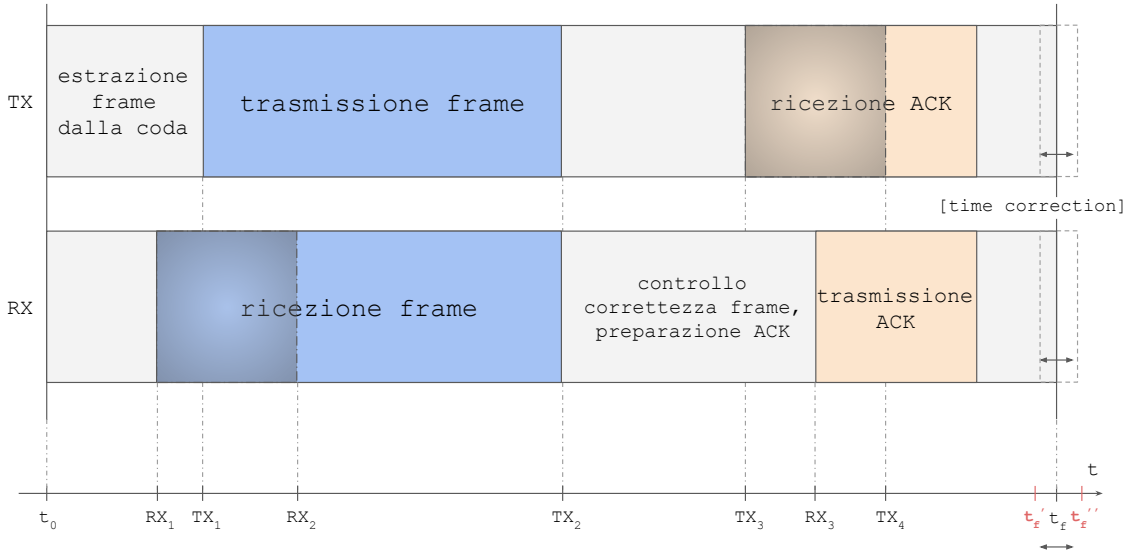


Figura 1.6: Timer interni al timeslot

Il nodo N_{TX} viene risvegliato al tempo t_0 e, dopo aver verificato la presenza di un frame nella coda destinata a N_{RX}, configura la propria antenna sulla frequenza stabilita e comincia la trasmissione (TX_1). Dopo un certo intervallo di tempo – proporzionale alla dimensione del pacchetto – all'istante TX_2 il nodo termina la trasmissione, disattiva la propria antenna e si prepara alla ricezione dell'eventuale frame di ACK (TX_3), riattivando l'antenna in modalità di ricezione. Se entro il tempo TX_4 non viene ricevuto alcun bit il nodo N_{TX} notifica la perdita dell'ACK ai livelli superiori dello stack e termina la propria routine. In caso di ricezione del messaggio di conferma, invece, manda un segnale di avvenuta trasmissione del frame di dato e, in presenza di un valore di time correction all'interno dell'ACK – e solo se proveniente da un *time source neighbor* – applica una correzione al proprio timer che porta ad una riduzione (t'_f) o aumento (t''_f) del timer che segna la conclusione dello slotframe. In alcune ottimizzazioni, N_{TX} è in grado di limitare al massimo i consumi se rileva l'assenza di pacchetti nella coda.

Il nodo N_{RX}, invece, si risveglia in t_0 e si pone in fase di ascolto all'istante RX_1. In modo analogo a quanto visto prima, se in RX_2 nessun bit viene ricevuto può spegnere anticipatamente l'antenna e terminare il proprio ciclo. In entrambi i casi il tempo di attesa prende il nome di *guard time*. A differenza del nodo che trasmette, però, N_{RX} è costretto ad un dispendio di energia proporzionale al guard time dato che non può riconoscere l'assenza di pacchetti nella coda del nodo trasmettitore. Il fenomeno prende il nome di *idle listening* e numerose soluzioni sono state proposte per ridurne al minimo l'incidenza sul consumo totale di energia. Nel caso in cui il traffico della rete sia periodico (o comunque prevedibile) la tecnica denominata Proactive Reduction of Idle Listening (PRIL) permette di ottenere notevoli risparmi energetici, riducendo significativamente il fenomeno dell'idle listening [26, 27]. Se l'informazione è correttamente ricevuta dall'antenna, all'istante TX_2 il nodo può analizzare il frame ricevuto e rilevare eventuali errori. Il frame di ACK (si ricordi, di lunghezza fissa) viene trasmesso e, se necessario, viene applicata una time correction di tipo frame based. In caso di corruzione del pacchetto, invece, il messaggio di conferma non viene inviato e il ciclo termina (anche qui, eventualmente dopo la time correction).

Schemi più completi [16] delineano gli orizzonti temporali delle varie primitive legate alla gestione della coda di pacchetti, alla comunicazione con il radiotrasmettitore, all'applicazione della correzione dei timer interni e così via. Opzionalmente, il nodo trasmettitore può effettuare un'operazione di Clear Channel Assessment (CCA) finalizzata a stabilire se il canale è già occupato da altre trasmissioni.

Tutte le operazioni devono essere svolte nei limiti della durata di un timeslot ($t_{TS} = 20\,\text{ms}$ in OpenWSN). In particolare, N_{RX} prima di trasmettere l'ACK può dover decomprimere, deframmentare e/o decifrare e autenticare il pacchetto, per cui l'utilizzo di appositi moduli hardware può garantire una drastica riduzione dei tempi e del consumo energetico.

1.3 Stack protocollare

Lo standard TSCH propone un metodo di accesso al canale affidabile, veloce ed orientato al risparmio energetico. Per rendere effettivo l'utilizzo delle reti WSN in ambiti reali una valida scelta consiste nell'adattare lo stack protocollare soprastante per consentire la segnalazione, la gestione e la comunicazione dei nodi. Di seguito vengono descritti una serie di protocolli che consentono ai dispositivi di integrarsi in reti IP esistenti e di comunicare con host e server remoti. Lo scambio di informazioni avviene attraverso appositi campi all'interno dell'header del frame IEEE 802.15.4, principalmente l'Information Element (IE) e l'Auxiliary Security Header. In particolare, l'IE può contenere informazioni aggiuntive di varia natura, poste direttamente nell'header (Header IE, opzionalmente autenticato) o nel payload (Payload IE, opzionalmente autenticato e cifrato) del frame. L'Auxiliary Security Header, invece, specifica gli algoritmi e i parametri di sicurezza utilizzati. In Figura 1.7 viene schematizzato un esempio, non esaustivo, di stack protocollare in cui vengono riportati i diversi protocolli, descritti nelle sezioni successive.

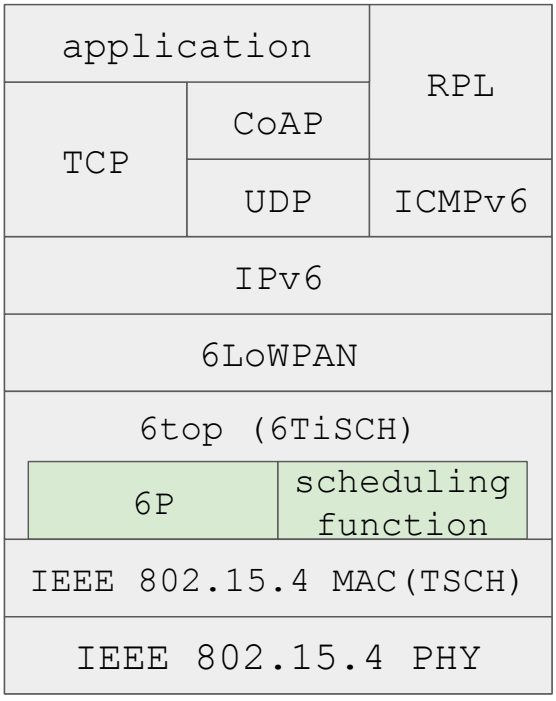


Figura 1.7: Esempio di stack protocollare

1.3.1 6TiSCH

IPv6 over the TSCH (6TiSCH) [28] è uno standard che descrive una configurazione minimale che fornisce connettività IPv6 ai nodi di una rete TSCH, ponendosi ad un livello intermedio tra il livello MAC e il livello di rete. L'RFC 8180 consiglia l'utilizzo di una cella di tipo `SHARED` in posizione $(0, 0)$, attiva, dunque, sia in ricezione che in trasmissione in modalità slotted aloha. I nodi già collegati alla rete utilizzano questa cella per inviare periodicamente dei messaggi *broadcast*, detti Enhanced Beacon (EB). Tali messaggi, trasmessi a livello *data-link*, non vengono confermati da un ACK e contengono una serie di IE che consentono ai nodi appena connessi di recuperare le informazioni sulla rete necessarie per effettuare il *join*. In particolare, il TSCH Synchronization IE contiene informazioni di sincronizzazione, quali l'ASN attuale e la *Join Metric* del nodo trasmettitore (rappresentazione del suo *rank* RPL, Sezione 1.3.3), mentre nel Channel Hopping IE è contenuto un identificativo a partire dal quale si ricava la *hopping sequence* utilizzata dalla rete per la selezione del canale logico. Un generico nodo, durante la fase di collegamento alla rete, si pone in ascolto di tali EB e, in una prima fase, li utilizza per la time synchronization (che in seguito sarà garantita dallo sfruttamento del traffico dati stesso o da messaggi di KA). Dopo aver ricevuto un certo numero di EB, effettua la selezione del proprio *parent node* basandosi sul valore più basso di *Join Metric* (Figura 1.8). Si costruisce, così, una topologia di nodi ad albero con una profondità limitata e, di conseguenza, il numero di *link* da attraversare durante una generica comunicazione è ridotto, con benefici in termini di latenza. Le metriche tipicamente utilizzate sono relative alla lunghezza, alla latenza o all'affidabilità dei link.

Il nodo padre selezionato, che in questa fase viene denominato Join Proxy (JP), fornisce una connettività di base che può essere estesa attraverso l'utilizzo di una funzione di scheduling minimale per la connessione dei nodi (*MSF*) [29] e un meccanismo per lo scambio coerente di tale schedule (*6P*) [30], entrambi parte del protocollo 6top.

6P Protocollo che definisce una serie di comandi utilizzati per la gestione delle celle schedulate tra una coppia di nodi. Durante la fase di *join* iniziale viene riservata una sola cella – detta *hard cell*, in quanto non rimovibile – all'interno dello slotframe per ogni comunicazione unidirezionale. Un generico nodo trasmettitore mantiene delle statistiche aggiornate sulla percentuale di effettivo utilizzo delle celle schedulate per ogni destinazione e, ad intervalli regolari, può decidere di richiedere un incremento (`ADD`) o decremento (`DELETE`) del numero delle stesse. Gli altri comandi supportati dal protocollo sono `COUNT`, `LIST`, `RELOCATE`, `CLEAR` e `SIGNAL`. Quest'ultimo consente lo scambio di messaggi generici tra nodi, la cui semantica dipende dall'implementazione specifica, e può essere utilizzato, ad esempio, per aggiornare il set di canali logici utilizzati per la trasmissione (Sezione 1.4). I messaggi 6P vengono incapsulati in appositi Payload IE e trasmessi, cifrati ed autenticati,

al nodo destinatario, direttamente raggiungibile tramite comunicazione *data-link*. Le richieste possono essere effettuate attraverso transazioni a 2 passi, nelle quali il nodo richiedente fornisce un insieme di celle candidate, o a 3 passi, nelle quali è il nodo ricevitore a proporle e ad attendere la selezione da parte del nodo richiedente. In ogni caso, le richieste vengono inoltrate a fronte di una decisione presa da una Scheduling Function (SF).

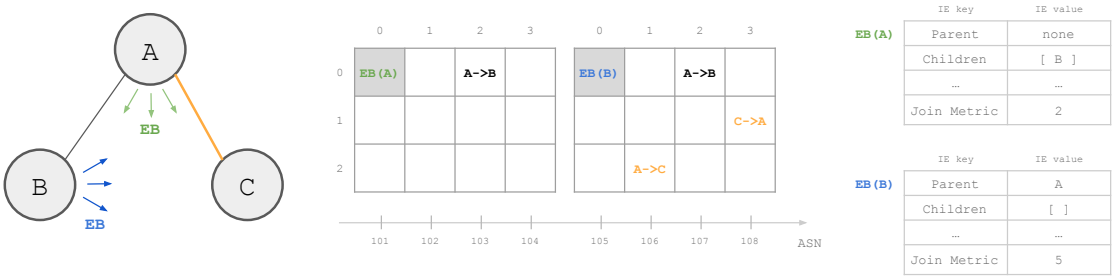


Figura 1.8: Scheduling statico minimale: connessione alla rete fornita dal JP, selezionato in base al valore più basso di Join Metric. A destra sono rappresentati due esempi verosimili di EB da cui il nodo C apprende le informazioni minimali sullo schedule. In seguito alla ricezione di 2 EB il nodo A viene scelto come JP ($ASN = 105$) e vengono schedulate due celle per garantire la comunicazione bidirezionale

Minimal Scheduling Function (MSF) Funzione di scheduling minimale proposta nell'ambito del protocollo 6top a complemento di 6P. Estende lo schedule statico fornito ai nodi in fase di *join* alla rete, mantenendo un numero di celle schedulate proporzionale alle necessità di traffico del link. Vengono definite delle finestre temporali di 16 slotframe, al termine delle quali la MSF calcola la percentuale di effettivo utilizzo delle celle schedulate. Se il nodo trasmettitore ritiene che la capacità di traffico richiesta dall'applicativo sia maggiore rispetto al *throughput* offerto dal numero attuale di canali può sfruttare il protocollo 6P per effettuare una richiesta di nuove celle. In modo complementare, il numero di celle viene ridotto se il loro utilizzo è stato limitato (Figura 1.9). In ogni caso è garantita la presenza di almeno una cella nell'insieme e lo scambio è reso coerente per mezzo di opportune operazioni (ritrasmissione, quarantena, reset) in caso di rilevazione di errori.

1.3.2 6LoWPAN

La serie di specifiche IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) fornisce i meccanismi necessari alla trasmissione efficiente di pacchetti IPv6

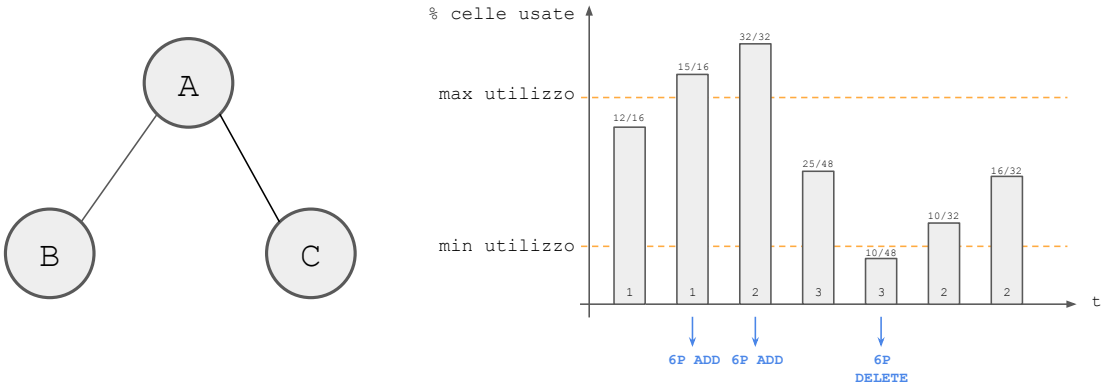


Figura 1.9: Scheduling dinamico delle celle: ogni 16 timeslot la MSF calcola la percentuale di utilizzo delle celle inviando dei messaggi di *6P ADD* al parent node in caso di valori sopra soglia e dei messaggi di *6P DELETE* in caso di valori sotto soglia. Il numero corrente di celle schedulate è indicato all'interno della barra verticale. È sempre garantita la presenza di almeno una cella

su reti IEEE 802.15.4, ponendosi ad un livello intermedio tra il MAC e il livello di rete. Le tre caratteristiche principali introdotte dallo standard sono la frammentazione, la *header compression* e il *layer-two forwarding* [31]. Il protocollo IPv6, data la sempre più crescente disponibilità di banda, prevede una Maximum Transmission Unit (MTU) di 1280 B. Tuttavia, questo valore è molto più elevato rispetto alla massima dimensione del frame supportata dallo standard IEEE 802.15.4, pari a 127 B. 6LoWPAN applica, dunque, una compressione dell'header di tipo *stateless*, che consiste nell'eliminazione dei campi dell'intestazione del pacchetto IPv6 che possono essere derivati a partire dalle informazioni di livello MAC presenti nell'header del frame IEEE 802.15.4. Spesso il traffico dati tra i nodi include delle informazioni ripetute, quali il prefisso della sottorete IPv6, l'identificativo del protocollo di livello superiore utilizzato e così via. L'algoritmo di compressione e decompressione, dunque, sfrutta la ridondanza di tali valori riducendo notevolmente le dimensione dell'header, che, in caso di comunicazione diretta su un singolo hop, può arrivare ad occupare solamente 7 B, informazioni relative al traffico UDP incluse.

Se la trasmissione è di tipo *link-local*, l'indicazione della sottorete di appartenenza dei due nodi viene totalmente omessa, in quanto derivabile dall'indirizzo IEEE 802.15.4 su 16 bit associato ai nodi. In questo contesto, dunque, la caratteristica di autoconfigurazione e di *neighbor discovery* del protocollo IPv6 gioca un ruolo fondamentale, in quanto l'indirizzo di rete è direttamente riconducibile all'indirizzo di livello 2. In caso di comunicazioni con l'esterno, invece, il numero di bit necessario a rappresentare la sottorete remota viene comunque ridotto attraverso l'utilizzo di

un Context ID, che associa un identificativo univoco alle sottoreti esterne note.

L'altra tecnica largamente utilizzata è la frammentazione dei pacchetti, necessaria a soddisfare i vincoli di MTU espressi da TSCH. I pacchetti IPv6 vengono dunque suddivisi in più frame *link-level* e i frammenti vengono inviati singolarmente attraverso la rete e ricostruiti lato ricevitore. 6LoWPAN utilizza una tecnica di deframmentazione virtuale dei pacchetti: in poche parole i frammenti ricevuti dai nodi intermedi vengono subito inoltrati senza attendere le restanti parti del pacchetto originale, che viene ricostruito direttamente sul nodo destinatario. I dispositivi, infatti, hanno di solito limitate risorse e, in particolare, insufficiente memoria per consentire la memorizzazione di tutti i frammenti. In questo modo, inoltre, si riducono le latenze end-to-end della comunicazione.

1.3.3 RPL

La connettività tra nodi sparsi all'interno della rete è garantita dalle regole di routing stabilite dal protocollo IPv6 Routing Protocol for Low-Power and Lossy Networks (RPL) [32]. Tale protocollo, di tipo *distance vector*, permette la costruzione di una topologia ad albero attraverso lo scambio di appositi messaggi. La costruzione della topologia è regolata da tecniche di minimizzazione dei consumi energetici e riconoscimento di eventuali incongruenze in essa. RPL sfrutta il protocollo ICMPv6 per la distribuzione di tali frame di configurazione e garantisce un routing efficiente e privo di cicli per pattern di rete Point-to-Multipoint (P2MP) e Multipoint-to-Point (MP2P).

A differenza della MSF, che fornisce connettività a livello MAC, RPL crea una topologia di nodi connessi su rete IPv6. La struttura che si forma è un DODAG (grafo aciclico orientato), memorizzato a livello globale nel nodo radice. Sono anche possibili tecniche di gestione distribuita del grafo, in genere difficilmente implementabili su dispositivi con ridotta disponibilità di risorse.

Lo standard prevede la definizione di un'apposita *objective function* che stabilisce la metrica di distanza relativa tra nodi da utilizzare. Alcuni esempi di tale metrica, detta anche *rank*, sono il consumo energetico stimato dei nodi e l'ETX. Nel primo caso, i nodi forniscono una stima del proprio livello residuo di energia, che assume un valore massimo in caso di nodo collegato costantemente ad una fonte di alimentazione o, in alternativa, calcolato in base al livello di energia residuo della batteria che lo alimenta. La metrica ETX, invece, approssima il numero atteso di trasmissioni previste affinché un pacchetto di dato raggiunga il nodo radice del DODAG. Il valore minimo di ETX è pari a 1 in caso di nodo direttamente collegato al *root node* su un link che presenta un basso tasso di errori di trasmissione. Il *rank*, a prescindere dal criterio con cui viene calcolato, è utilizzato dai nuovi nodi per la selezione del *parent node* all'interno dell'albero.

Il protocollo RPL definisce quattro categorie di messaggi di controllo: DODAG Information Solicitation (DIS), DODAG Information Object (DIO), Destination Advertisement Object (DAO), DAO-ACK.

DIO Rappresenta la fonte principale di informazione relativa alla posizione di un generico nodo all'interno dell'albero topologico. All'interno di questo messaggio si trovano le indicazioni relative al *rank*, all'identificativo del nodo trasmettitore e all'istanza del DODAG RPL. È possibile, infatti, costruire numerose topologie di nodi, eventualmente sovrapposte tra loro. Questo messaggio ha origine nel nodo root e viene in seguito propagato ai nodi circostanti. I dispositivi che ricevono un DIO eseguono un algoritmo per la selezione del miglior *parent node* e, allo stesso tempo, calcolano il proprio valore di *rank*, ottenuto sommando il valore ricevuto dal nodo trasmettitore al peso del link che li collega con quest'ultimo. Se il nodo ricevitore è un nodo *router* il messaggio DIO aggiornato viene inoltrato ai nodi sottostanti. Attraverso questo scambio di informazioni è possibile garantire l'*upward routing*, che permette ai nodi foglia di trasmettere delle informazioni in direzione del nodo radice, talvolta denominato *gateway*.

Il *rate* di trasmissione dei messaggi DIO è proporzionale alla dinamicità della rete: in caso di link stabili i messaggi di controllo saranno limitati, mentre il ritmo sarà più elevato in caso di modifiche frequenti alla topologia.

DAO Messaggio che abilita il supporto al traffico P2MP e consente di confermare le scelte di routing effettuate dai *leaf node*. I DAO sono trasmessi, dunque, in direzione *upward* in seguito alla ricezione di uno o più messaggi DIO. Anche in questo caso all'interno del frame sono presenti informazioni riguardanti il DODAGID (identificativo del nodo all'interno dell'albero), l'istanza RPL in questione e, in aggiunta, un flag `'K'` di richiesta di conferma e una lista estendibile di informazioni ulteriori. Nella modalità operativa *non-storing* – l'unica implementata da TSCH – i nodi utilizzano la tecnica del *source routing*, che prevede la propagazione della lista di *parent node* fino alla radice del DODAG. Ricevute tali informazioni, il nodo root è in grado di allestire una tabella di routing globale: tutte le trasmissioni provenienti dai nodi foglia, infatti, verranno inoltrate dapprima in direzione del nodo radice e poi instradate verso la destinazione finale. I nodi intermedi si limitano a ricevere i messaggi DAO dai nodi sottostanti, aggiornare la lista di nodi presente all'interno del campo opzionale del frame – includendo se stessi come *parent node* del nodo da cui hanno ricevuto il messaggio – e inoltrare, a propria volta, il messaggio DAO in direzione del nodo radice.

DIS Messaggio di sollecitazione che consente ad un nodo di richiedere anticipatamente la trasmissione di un DIO da parte dei nodi vicini.

DAO-ACK Messaggio opzionale di avvenuta ricezione del DAO che conclude la transazione. Da questo momento il nodo è connesso alla rete e può, a sua volta, inviare dei messaggi DIO periodici. La trasmissione di tale messaggio è vincolata alla presenza del flag di conferma `'K'` all'interno del DAO corrispondente.

1.3.4 CoAP

La finalità delle reti WSN è quella di rendere raggiungibili in rete un certo numero di dispositivi, spesso dotati di sensori, che dispongono di limitate risorse computazionali. La maggior parte del traffico nella rete Internet è generata da richieste HTTP che vengono trasportate dal protocollo TCP. La natura connection oriented del protocollo, alla base delle dispendiose operazioni di controllo della congestione del traffico e di gestione delle ritrasmissioni, ne rende impossibile l'utilizzo nelle comunicazioni in reti TSCH. Constrained Application Protocol (CoAP) è stato implementato con lo scopo di emulare le caratteristiche del protocollo HTTP, selezionando un *subset* delle sue funzionalità e rendendolo più efficiente dal punto di vista energetico [33]. La sua applicazione principale è l'IoT e, analogamente al protocollo HTTP, è basato su uno schema client-server secondo il quale un nodo, che assume il ruolo di client, richiede l'applicazione di un'azione (identificata dal `metodo`) nei confronti di una risorsa (identificata da un `URI`) presente sul nodo server. A differenza di HTTP, però, lo scambio di messaggi (richiesta ed eventuale risposta) avviene in modalità asincrona e non affidabile attraverso l'utilizzo del protocollo di trasporto UDP: in questo modo le ritrasmissioni non vengono gestite a questo livello ma delegate al protocollo TSCH.

L'informazione scambiata tra due *endpoint*, in reti WSN, è rappresentabile, tipicamente, su pochi byte (ad esempio la temperatura misurata all'interno di un locale). Ne consegue una limitata – se non assente – frammentazione introdotta dal protocollo 6LoWPAN, con conseguente vantaggio in termini energetici e di latenza. In particolare, l'header di un messaggio CoAP ha una lunghezza fissa di 4 B e contiene, principalmente, un `Message ID` univoco per la rilevazione di duplicati ed un flag che indica se il messaggio necessita di essere confermato (`CON`) o meno (`NON`). Nel primo caso, i messaggi vengono ritrasmessi finché il ricevitore non riceve un ACK CoAP contenente lo stesso identificativo. L'utilizzo di UDP implica la possibilità di raggiungere più destinazioni contemporaneamente (con una singola richiesta CoAP), attraverso l'utilizzo di indirizzi IPv6 multicast. Inoltre, analogamente a HTTP, esiste una versione sicura del protocollo (CoAPs), basata su DTLS.

Il protocollo segue il paradigma REpresentational State Transfer (REST), introdotto in HTTP e incentrato sulla definizione di risorse (fonte dell'informazione) consultabili e modificabili attraverso richieste di tipo *stateless*. I metodi supportati sono `GET`, `PUT`, `POST` e `DELETE` e sono riconducibili agli equivalenti in HTTP. Il metodo `GET` restituisce al client la rappresentazione attuale della risorsa ed è l'unico metodo

che possiede entrambe le proprietà di sicurezza (non vengono eseguite operazioni collaterali in corrispondenza di una richiesta di questo tipo) e di idempotenza (il risultato ottenuto è lo stesso anche a fronte di richieste ripetute). Attraverso il metodo `PUT` (idempotente) è possibile richiedere la creazione o, più appropriatamente, la modifica di una risorsa identificata per mezzo di un `URI` esplicitamente fornito al server. Il metodo `POST`, con qualche leggera differenza, provoca l'esecuzione di una determinata routine da parte del server ed è utilizzato per inserire nuove risorse (le cui rappresentazioni sono contenute nel corpo del messaggio) all'interno del sistema. La proprietà di idempotenza non è garantita in questo caso. Per ultimo, attraverso una richiesta `DELETE`, è possibile richiedere la rimozione della risorsa (idempotente, poiché la cancellazione di una risorsa inesistente non genera errori sul server).

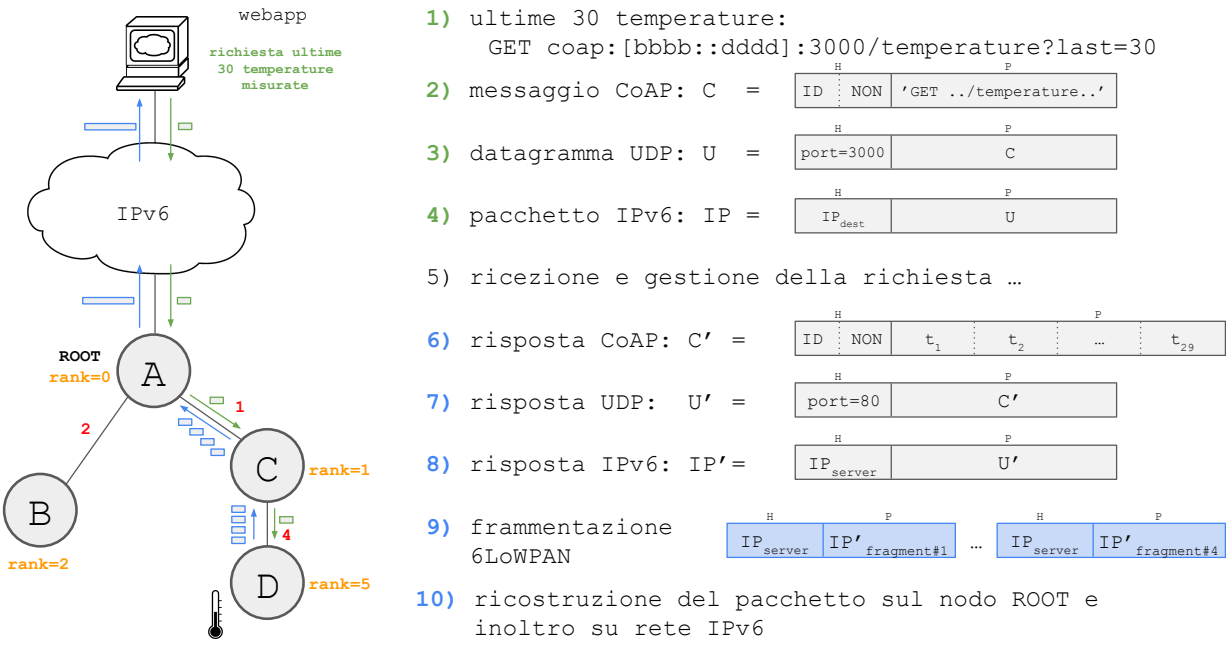


Figura 1.10: Interazione CoAP client-server con frammentazione 6LoWPAN. I valori in rosso rappresentano il peso di ciascun link, da cui deriva il *rank* calcolato dai nodi

Come esempio, il nodo coordinatore della rete può sollecitare una misurazione da parte di un nodo sensore per mezzo della richiesta GET così composta: `coap://[bbbb::12:4b00:18e0:ba23]:61620/temperature`. CoAP, infine, consente ad un nodo di scoprire le risorse che un server ha reso accessibili attraverso una richiesta `GET` preliminare indirizzata all'`URI` contenente il prefisso `.well-known` (`coap://[bbbb::12:4b00:18e0:ba23]:61620/.well-known/sensors`, seguendo l'esempio precedente, potrebbe restituire `'/temperature, /humidity, /pressure'`).

In Figura 1.10 viene raffigurata l'interazione CoAP tra i nodi di una rete WSN, la cui topologia RPL è costruita a partire dai valori di *rank* raffigurati, ed un server remoto raggiungibile attraverso la rete IPv6.

1.4 Blacklisting e Whitelisting

Nelle sezioni precedenti si è citata la possibilità di apportare delle ottimizzazioni alla funzione di estrazione della frequenza da utilizzare per la comunicazione, nell'ottica di ottenere, a lungo termine, probabilità di errore sulla trasmissione del pacchetto più ridotte con conseguente aumento dell'affidabilità e del tempo di vita del sistema (quest'ultimo inversamente proporzionale al consumo energetico). Per fare ciò, una possibilità è quella di limitare le trasmissioni ad un sottoinsieme di canali che presentano una bassa probabilità di errore: supponendo di avere a disposizione i canali $\mathcal{C} = \{ch_1, ..., ch_n\}$, la selezione in Equazione (1.1) avviene su $\mathcal{W} \subseteq \mathcal{C}$ in caso di whitelisting o, in modo complementare, sull'insieme $\mathcal{C} \setminus \mathcal{B}$ in caso di blacklisting. Nel seguito vengono analizzate esclusivamente tecniche di whitelisting, poiché permettono di effettuare in maniera diretta la selezione dei canali migliori. Le rappresentazioni, tuttavia, sono equivalenti ed è possibile esprimerle in ogni istante nella forma complementare notando che $\mathcal{W} \cup \mathcal{B} = \mathcal{C}$ e $\mathcal{W} \cap \mathcal{B} = \emptyset$. Supponendo di poter determinare ed utilizzare in ogni istante il sottoinsieme di canali $\mathcal{W}$ ottimo, cosa realisticamente improbabile, si può dedurre che la qualità complessiva della comunicazione migliori rispetto all'utilizzo dell'intera gamma di canali a disposizione. Se $|\mathcal{W}| = 1$ le trasmissioni hanno sempre luogo sul canale migliore e si ottiene l'ottimo in termini di prestazioni, eludendo, di fatto, le funzionalità di channel hopping introdotte da TSCH.

Le tecniche di whitelisting cercano di approssimare periodicamente il vettore di canali ottimo, selezionando le frequenze in base all'andamento medio di una determinata metrica di qualità della trasmissione. I criteri di selezione possono essere fondati sull'analisi di parametri base come il Received Signal Strength Indication (RSSI), il PDR, la latenza media misurata sulle trasmissioni effettuate sul canale oppure – utilizzando tecniche più avanzate – sulla rilevazione di interferenze durante celle appositamente schedulate (ATSCH [34]) o durante i primi istanti di ogni timeslot (ETSCH [35]). Le tecniche basate su RSSI si sono dimostrate non ottime, in quanto tale valore è a disposizione dei nodi solamente quando il frame (di dato o di ACK) viene correttamente ricevuto e decodificato. In caso di perdita o corruzione dei frame, infatti, l'RSSI non viene calcolato, rappresentando, di fatto, la potenza del segnale ricevuto e non il tasso di perdita di informazione. La metrica più largamente utilizzata è il PDR, calcolato indipendentemente dai nodi per ciascun link su cui essi comunicano. In particolare, la presenza di reti Wi-Fi nelle immediate vicinanze potrebbe introdurre delle interferenze sistematiche, con

conseguente incremento del numero di pacchetti persi sui canali coinvolti dall'interferenza stessa (ogni canale IEEE 802.11 può interferire approssimativamente con quattro canali logici IEEE 802.15.4, Figura 1.5). Si può supporre, tuttavia, una buona affidabilità media per comunicazioni a corto raggio ($< 3\,\mathrm{m}$). È dimostrato che i risultati migliori si ottengono mediante l'utilizzo di whitelist locali ai singoli link di comunicazione [36]. Questo perché la distribuzione spaziale del sistema può essere ampia e, quindi, la disattivazione globale di una determinata frequenza potrebbe migliorare le prestazioni nella zona maggiormente colpita dall'interferenza ma, allo stesso tempo, potrebbe ridurre il *throughput* complessivo della rete, e lo stesso canale potrebbe avere una buona qualità in altre porzioni della rete.

Un altro aspetto da considerare è la possibilità di effettuare delle trasmissioni simultanee. La matrice TSCH, infatti, consente di schedulare fino a N_{ch} comunicazioni unidirezionali tra altrettante (disgiunte) coppie di nodi. La corretta trasmissione dei frame è garantita solo se i nodi condividono la stessa permutazione dei canali logici H, la stessa funzione di estrazione dei canali e i valori di $|H|$ e N_{slots} sono primi tra loro. Se uno o più di questi valori differiscono tra le coppie di nodi è possibile che si verifichino delle sovrapposizioni nelle trasmissioni, con conseguenti peggioramenti in termini di consumi energetici e latenze. Le inconsistenze possono derivare da una diversa selezione dei canali ($perm_1(\mathcal{W}) = H_1 \neq H_2 = perm_2(\mathcal{W})$) oppure da una diversa implementazione della funzione di selezione del canale (Capitolo 3). Tuttavia, la necessità di effettuare trasmissioni simultanee si presenta solo in determinate condizioni di elevata densità di nodi, mentre in reti di dimensione ridotta è spesso possibile schedulare una sola trasmissione per timeslot.

Di particolare rilievo per le tecniche discusse in questo lavoro di tesi è l'implementazione della tecnica di whitelist *link-based* fornita da LABeL [37]. L'insieme $\mathcal{W}$ viene calcolato periodicamente ed include un numero di canali variabile, dipendente dalla probabilità media di successo della comunicazione sugli stessi. Nel dettaglio, un generico nodo calcola, per ogni nodo vicino N, il PDR relativo a ciascun canale ch attraverso:

$$PDR\,(N,\ ch) = \frac{\#ACK_{\ [N,\ ch]}}{\#TX\ _{[N,\ ch]}} \tag{1.2}$$

A partire dal valore più elevato viene definita una soglia PDR_{th} che consente di escludere i canali peggiori da $\mathcal{W}$, garantendo la presenza di almeno 3 canali ($|\mathcal{W}| \geq 3$). Tale whitelist viene consultata al momento dell'estrazione della frequenza (Equazione (1.1)) e la comunicazione ha luogo solamente se $ch \in \mathcal{W}$. In caso contrario si possono intraprendere due azioni: con probabilità p si decide di utilizzare comunque il canale ch, reinserendolo nella whitelist per verificare se è ancora soggetto ad interferenze, mentre con probabilità $1-p$ i due nodi selezionano

un secondo canale appartenente alla whitelist ($ch' = rand(\mathcal{W})$[2]) su cui trasmettere. L'implementazione garantisce l'estrazione dello stesso valore di channel offset ai due estremi della comunicazione. Se ciò non fosse, i nodi non sarebbero in grado di comunicare in quanto operanti su bande di frequenza disgiunte.

In Figura 1.11 viene riportato un esempio di funzionamento dell'algoritmo su due finestre temporali. Nella prima (1) si aggiorna la whitelist $\mathcal{W}$ escludendo i canali che hanno performato peggio (in questo caso il solo canale $ch = 4$). Nella finestra temporale successiva si suppone di voler trasmettere un frame di dato dal nodo A al nodo B e, dunque, si valutano i tre scenari possibili. Se il canale logico estratto per la comunicazione è $ch \in \mathcal{W}$ il frame può essere trasmesso immediatamente ($2a$). In caso opposto, con probabilità p si reinserisce il canale in precedenza rimosso dalla whitelist e la comunicazione ha luogo proprio su ch ($2b$), mentre con probabilità $1 - p$ si estrae il nuovo canale da utilizzare, lasciando invariato $\mathcal{W}$ ($2c$).

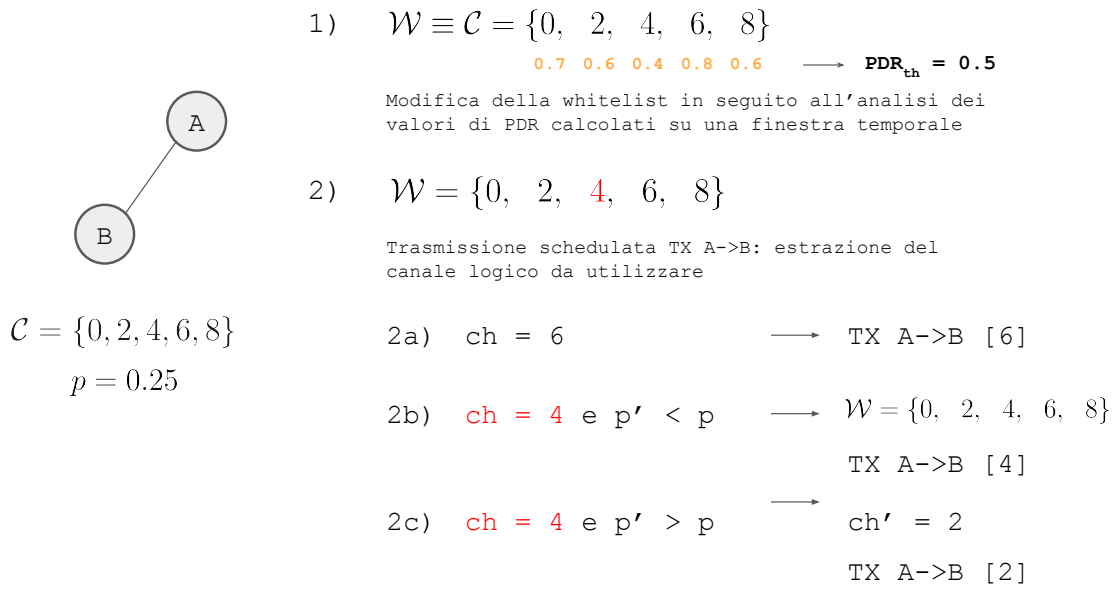


Figura 1.11: Gestione della whitelist in LABeL, con $p' = rand(0,\ 1)$. In giallo i valori di PDR calcolati in relazione alla finestra temporale 1, per ogni canale $ch \in \mathcal{W}$. Il canale $ch = 4$ (in rosso) è escluso dalla whitelist all'istante 2.

È necessario garantire che N_{TX} e N_{RX} condividano la stessa whitelist $\mathcal{W}$ in modo che possano calcolare lo stesso canale logico ch da utilizzare per la trasmissione. In caso di incoerenza, Equazione (1.1) potrebbe estrarre canali diversi, precludendo la comunicazione tra i nodi. In LABeL lo scambio di informazioni riguardanti $\mathcal{W}$

[2]Canale estratto pseudo-casualmente dalla whitelist

avviene per mezzo di messaggi 6P (Sezione 1.3.1), in particolare ponendo la lista di canali attivi nell'IE del frame. Tali messaggi vengono scambiati all'interno della cella condivisa (0, 0), che può essere soggetta ad interferenze in caso di trasmissioni sovrapposte da parte di altri nodi. Vengono dunque mantenute due whitelist, quella condivisa con il nodo ricevitore ($\mathcal{W}$) e quella non ancora confermata da un messaggio di ACK ($\mathcal{W}_{tmp}$). Nel descrivere il protocollo, gli autori considerano trascurabile la perdita dei messaggi di ACK durante le fasi di scambio della whitelist. Questa assunzione è, tuttavia, troppo restrittiva e non trova riscontro nei dati acquisiti in un contesto di WSN reale e utilizzati come riferimento per questo lavoro di tesi, dai quali si ricava un errore sul frame pari a $\epsilon_f = 12.6\%$ e una perdita dell'ACK pari a $\epsilon_{ACK} = 8\%$ [38].

Nel Capitolo 2 viene proposto un algoritmo di scambio coerente dell'informazione tra due nodi finalizzato a garantire la consistenza della whitelist, mentre in Capitolo 3 viene presentata una tecnica di whitelisting probabilistico – sviluppata in maniera indipendente rispetto a LABeL – caratterizzata da un'osservazione continua dello stato del canale e da una conseguente modifica graduale della probabilità di estrazione del singolo canale.

Capitolo 2

Protocollo di consistenza

2.1 Introduzione

Come visto nelle sezioni precedenti, la coerenza delle informazioni necessarie alla scelta del canale su cui trasmettere è un requisito fondamentale per la corretta comunicazione tra due nodi. In particolare, è possibile identificare due informazioni principali: la permutazione di canali H e la *hopping function* ν. La prima consiste in una lista ordinata di canali selezionati a partire dai canali logici $\mathcal{C} = \{ch_1, ..., ch_n\}$ a disposizione (eventualmente un sottoinsieme degli stessi), opportunamente posizionati in modo tale da garantire che l'effetto di un'interferenza non influisca sulla ritrasmissione successiva. La funzione di selezione del canale ν, invece, riceve come parametro il vettore H ed estrae il canale logico da utilizzare, a partire dai valori correnti di ASN e channel offset. La hopping function in Equazione (1.1), definita nello standard TSCH, ne è un esempio. Gli algoritmi di whitelisting eseguono periodicamente delle operazioni di monitoraggio dello stato dei canali al termine delle quali, se necessario, vengono aggiornati i parametri per la trasmissione tra una specifica coppia di nodi. Le modifiche possono riguardare sia H che ν, a seconda del tipo e dell'implementazione dell'algoritmo. In questo capitolo, per semplicità, il termine ν viene utilizzato per indicare una generica configurazione dei criteri di selezione del canale espressa in termini di hopping function o di permutazione dei canali. L'operazione di scambio dell'informazione ν, denominata ν-*exchange*, comincia nell'istante in cui il nodo trasmettitore N_{TX} produce un nuovo valore di ν e termina nel momento in cui entrambi i nodi sono in grado di stabilire che l'aggiornamento sia stato effettuato in modo coerente.

Nel contesto delle reti di calcolatori esistono numerosi protocolli che consentono a due nodi di instaurare una connessione, il più noto dei quali è sicuramente il protocollo TCP [39]. In questo standard viene introdotta una fase di segnalazione preliminare, detta *three way handshake*, che consiste nello scambio dei messaggi `SYN`, `SYN-ACK` e `ACK`. Il nodo client trasmette il messaggio `SYN` in direzione del server che, dopo aver allocato le risorse necessarie per la connessione, risponde con un

messaggio di SYN-ACK. La trasmissione del frame di ACK pone fine allo scambio. Se tutti i messaggi vengono inoltrati a destinazione, la fase di *handshake* può ritenersi conclusa e i nodi possono iniziare a comunicare attraverso la rete TCP/IP. Al contrario, la mancata ricezione di uno dei messaggi provoca l'interruzione della transazione dopo un certo intervallo di tempo stabilito da appositi timer schedulati su entrambi i nodi. In particolare il timer sul nodo N_{TX} rileva la perdita di SYN-ACK, mentre quello sul nodo N_{RX} rileva la perdita di ACK. In caso di mancata conclusione della transazione, tuttavia, l'unico effetto indesiderato è l'allocazione temporanea di risorse non effettivamente utilizzate e, ad ogni modo, il nodo client può decidere di ritentare la connessione in un secondo momento.

Una soluzione di questo tipo, purtroppo, non è applicabile nel contesto di ν-exchange in reti TSCH poiché la mancata conclusione della transazione provocherebbe un'incongruenza tra le informazioni ν memorizzate nei due nodi, con conseguenti possibili disconnessioni causate dall'utilizzo di frequenze diverse ai due estremi della comunicazione.

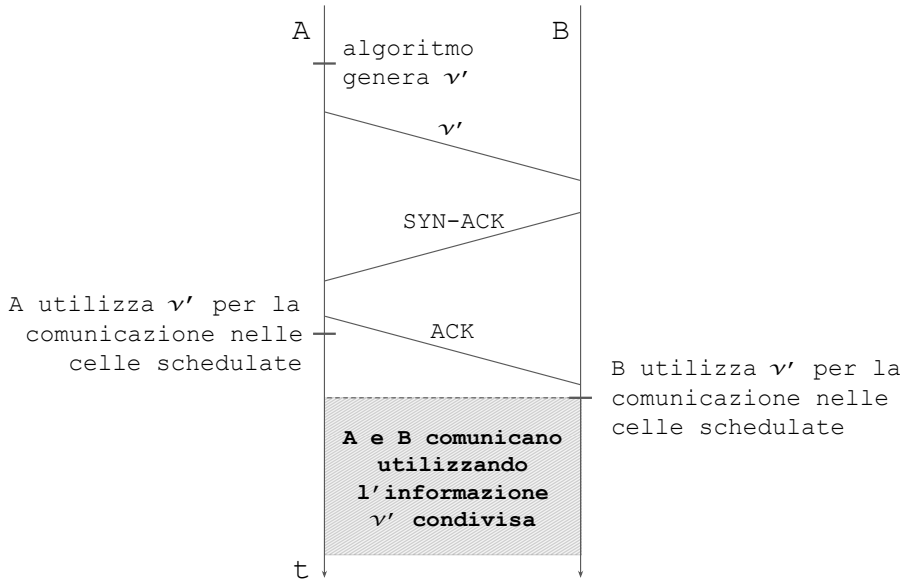
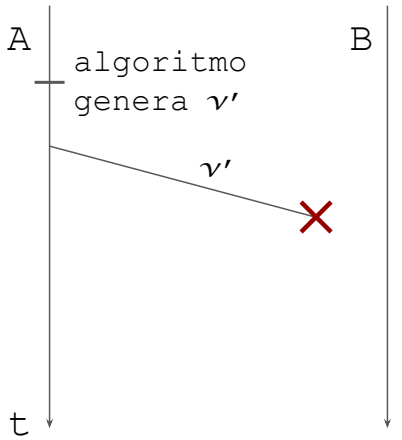


Figura 2.1: Schema di scambio di ν: aggiornamento correttamente effettuato

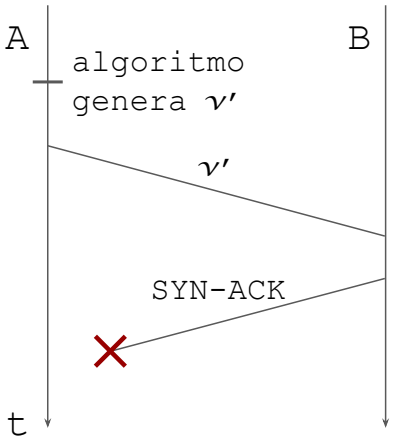
Di seguito si illustrano i possibili scenari di transazione a tre passi e si identificano le possibili incongruenze insieme ai requisiti di robustezza che un algoritmo di ν-exchange dovrebbe possedere. Negli esempi proposti si analizza lo scambio di informazione tra due nodi, A e B, che inizialmente comunicano per mezzo della configurazione definita da ν. Gli aggiornamenti successivi (ad esempio ν') hanno sempre origine nel nodo A, su cui opera un generico algoritmo di controllo dello stato del canale e di aggiornamento della configurazione, e sono propagati al nodo

B. Tutte le celle schedulate tra i due nodi, a prescindere dalla direzione della comunicazione, utilizzano l'informazione contenuta nel ν corrente. L'obiettivo principale è assicurare, in ogni istante, la connettività tra i due nodi garantendo che le celle schedulate utilizzino un valore di ν condiviso. Le fasi di scambio rappresentate in Figura 2.1 seguono lo schema del *three way handshake* del protocollo TCP. In seguito alla generazione di ν', il nodo A trasmette un frame contenente l'aggiornamento e si pone in uno stato di attesa del `SYN-ACK` corrispondente. Quando il messaggio `SYN-ACK` viene ricevuto, il nodo A riconosce che l'aggiornamento ha raggiunto il nodo destinatario e, a conclusione della transazione, trasmette a propria volta un `ACK` in direzione di B e modifica la propria configurazione secondo le informazioni contenute in ν'. Infine il nodo B, ricevuto l'ultimo `ACK`, aggiorna il proprio parametro ν, determinando la conclusione dell'algoritmo di scambio.

In uno scenario reale, tuttavia, gli errori di trasmissione sui link – specialmente se a lunga distanza o influenzati da interferenze esterne – non sono trascurabili, per cui uno o più messaggi dello scambio precedente possono non raggiungere il nodo destinatario. Dato che l'effettivo aggiornamento della configurazione avviene in seguito alla ricezione del secondo messaggio di `SYN-ACK`, la perdita del frame di aggiornamento (Figura 2.2a) o dell'`ACK` trasmesso da B (Figura 2.2b) non provoca inconsistenze tra i nodi. In questo caso, semplicemente, i valori di ν restano invariati e il nodo A può effettuare un nuovo tentativo di ν-exchange.



(a) Perdita del frame di aggiornamento



(b) Perdita del messaggio di ACK trasmesso da B

Figura 2.2: Schema di scambio di ν: perdita non critica di messaggi

In caso di perdita dell'ultimo frame di `ACK` (Figura 2.3), invece, i nodi si troverebbero in uno stato incoerente poiché A avrebbe già effettuato l'aggiornamento, mentre B, rilevando la perdita del messaggio, manterrebbe inalterata la propria

configurazione ($\nu' = \nu_A \neq \nu_B = \nu$). In questo caso la connettività non è più assicurata: data una generica cella ($A \rightarrow B$), schedulata in corrispondenza del channel offset c, è possibile che durante un tentativo di trasmissione si utilizzino due canali logici differenti ($\exists\ ASN : ch_A \neq ch_B$).

Occorre, dunque, perfezionare l'algoritmo di ν-exchange in modo da gestire i possibili errori di trasmissione. Gli aggiornamenti locali dell'informazione ν vengono effettuati in due momenti distinti e, in particolare, il nodo A non è in grado di stabilire se il nodo B abbia ricevuto o meno l'ultimo messaggio della transazione. Si potrebbe essere tentati dall'introdurre un ulteriore messaggio di ACK, dal nodo B al nodo A, che consenta di confermare la corretta ricezione del terzo frame dello scambio in Figura 2.1 e, di conseguenza, notificare al nodo A che l'aggiornamento è avvenuto anche su B. Questa soluzione, che introduce una ricorsività nello scambio di frame e relativi ACK, non consente di risolvere i problemi relativi all'incoerenza dell'informazione. In caso di errore di trasmissione su quest'ultimo pacchetto, infatti, si otterrebbe uno scenario in cui $\nu = \nu_A \neq \nu_B = \nu'$.

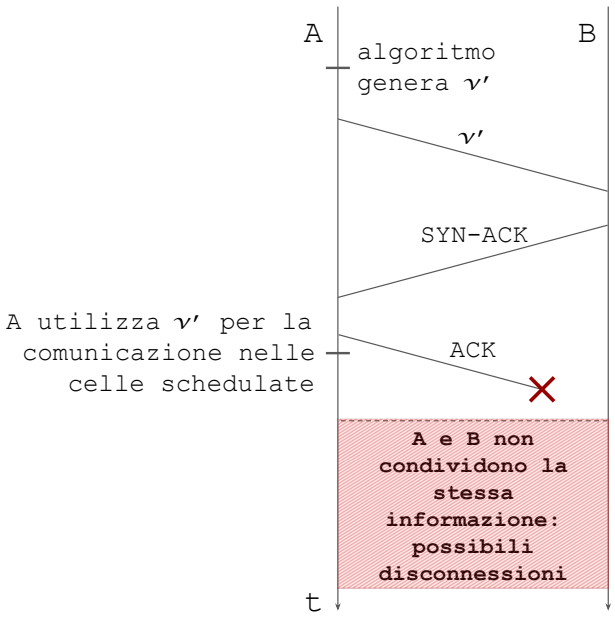


Figura 2.3: Schema di scambio di ν: perdita critica di messaggi

2.2 CONSIP

Il protocollo Consistency Protocol (CONSIP) consente di effettuare un ν-exchange coerente tra due nodi, utilizzando una tecnica di *three way handshake* più robusta. La coerenza viene garantita attraverso l'utilizzo di celle aggiuntive durante la

fase di scambio. Dopo aver effettuato il *join* alla rete, infatti, i nodi schedulano una cella secondaria C_{backup} per ogni cella C_{main} necessaria a soddisfare i requisiti di *throughput* espressi dall'applicazione. Le celle di backup vengono utilizzate esclusivamente durante le fasi di scambio di ν e hanno lo scopo di garantire la presenza di due canali di comunicazione distinti attivi in parallelo. In particolare, le celle C_{main} trasmettono frame secondo la configurazione ν, mentre le celle C_{backup} utilizzano il valore aggiornato ν'. I due canali di comunicazione restano attivi contemporaneamente fino alla conclusione dello scambio. Se il ν-exchange termina con successo, e dunque ν viene aggiornato con ν', le celle C_{backup} assumono il ruolo di celle principali e le celle C_{main} vengono disattivate. In questo modo, l'aumento dei consumi energetici introdotti dalla presenza di un numero doppio di celle è limitato alla durata del ν-exchange. In caso di fallimento della transazione, le celle C_{backup} vengono nuovamente disattivate e le trasmissioni continuano ad avere luogo in corrispondenza delle celle C_{main}, già attive in precedenza.

La Figura 2.4 illustra un semplice esempio di topologia di rete, che prevede la sola comunicazione unidirezionale $A \rightarrow B$ per mezzo di una singola cella schedulata e della relativa cella di backup. Il protocollo CONSIP non definisce i criteri per la scelta della posizione di C_{backup}: in questo caso l'indice di slot offset è stato selezionato in modo tale da rendere le due celle equispaziate su ripetizioni successive dello slotframe, mentre il channel offset è stato scelto in maniera pseudo-casuale.

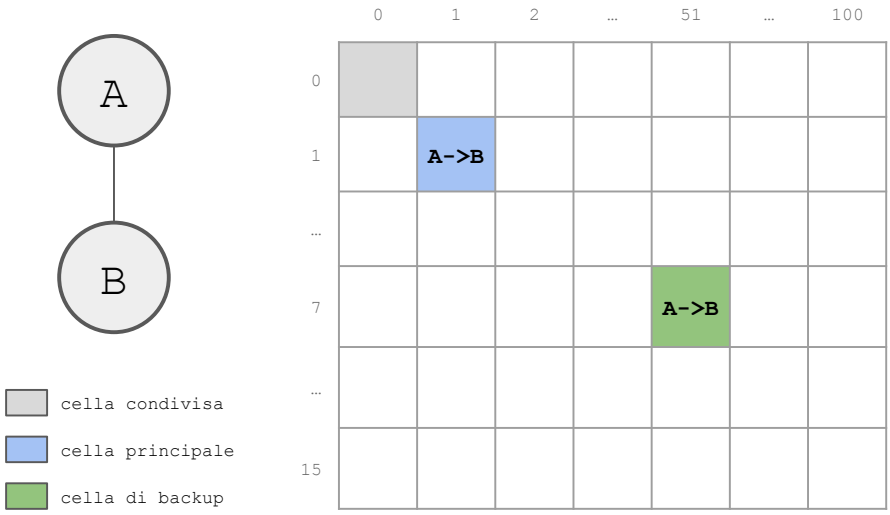


Figura 2.4: Esempio di slotframe nel protocollo CONSIP

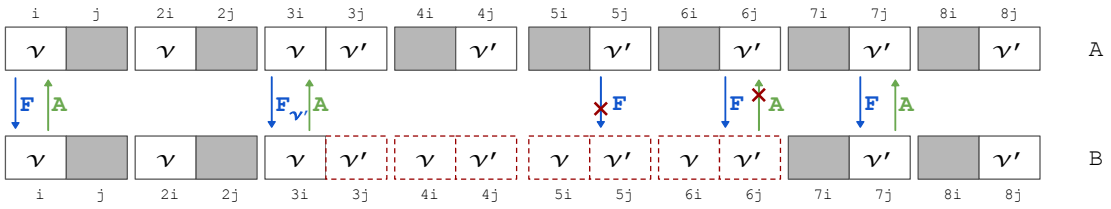
Affinché l'algoritmo possa essere supportato dai dispositivi, l'implementazione del protocollo di livello *data-link* presente sui nodi deve fornire le seguenti primitive: `is_main`, `swap` e `set_configuration`. Attraverso l'operazione `is_main(`C_x`)` si è in grado di stabilire se la cella C_x sia di tipo C_{main} o C_{backup}, mentre la primitiva

`swap(`C_x`,` C_y`)` consente di invertire il ruolo delle due celle passate come parametro. Va garantita, in ogni istante, la presenza di una sola cella principale e una sola cella secondaria per ogni coppia di celle presa in considerazione. Attraverso la funzione `set_configuration(`C_x`,` ν_x`)`, infine, è possibile aggiornare la configurazione per la generica cella C_x. Il secondo parametro ν_x, in particolare, può assumere il valore nullo `'-'`. In questo caso l'effetto è quello di disattivare la cella fino alla successiva istruzione di `set_configuration`.

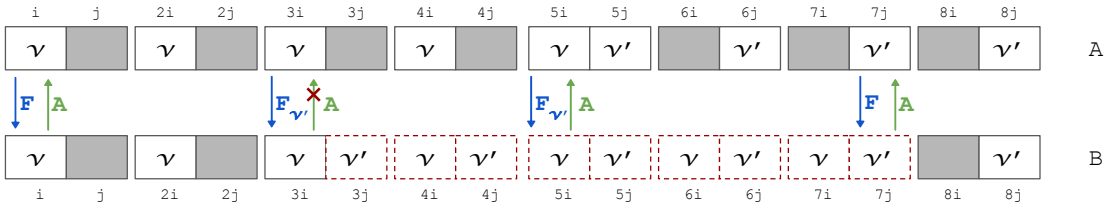
Il funzionamento dell'algoritmo è raffigurato in Figura 2.5. Le due celle schedulate per lo slotframe di riferimento sono identificate dagli indici i e j (in Figura 2.4 rispettivamente in posizione 1 e 51). Inizialmente la cella di backup viene disabilitata e la trasmissione dei frame di dato e di ACK avviene sulla cella C_{main} (in partenza identificata dall'indice i). All'istante $3i$ il nodo A ha a disposizione l'aggiornamento ν' e lo trasmette al nodo B all'interno di un apposito frame $\mathcal{F}_{\nu'}$, sfruttando, ad esempio, l'header Channel Hopping IE. Entrambi i nodi sono in grado di calcolare la stessa frequenza fisica da utilizzare per la trasmissione a partire dalla configurazione corrente della cella (indicata all'interno delle celle in figura). Sono possibili tre scenari per la trasmissione di $\mathcal{F}_{\nu'}$: `DATA-ACK`, `DATA-NOACK` e `NODATA`.

Nel primo caso (Figura 2.5a) il nodo B riceve l'aggiornamento e, come conseguenza, attiva le celle secondarie (`set_configuration(`j`,` ν'`)`). Da questo momento entrambe le celle si trovano in uno stato attivo. Ciò implica che il nodo B è in grado di ricevere frame, nei timeslot successivi, sia su C_{main} che su C_{backup}, configurando l'antenna sulle frequenze fisiche opportune, ricavate rispettivamente a partire da ν e ν'. Durante questa fase, detta di *double listening*, il consumo energetico del nodo B è quasi doppio. Allo stesso tempo, il frame di ACK inviato da B viene ricevuto correttamente dal nodo A, che può effettuare l'aggiornamento locale della propria configurazione. Nel dettaglio, la cella principale viene disabilitata ($4i$ e successive) e al suo posto viene attivata la cella C_{backup}, finora non attiva. Queste operazioni vengono effettuate rispettivamente attraverso le istruzioni `set_configuration(`i`, -)` e `set_configuration(`j`,` ν'`)`. I ruoli delle celle vengono, infine, invertiti (`swap(`C_{main}`,` C_{backup}`)`) e, da questo momento in poi, il nodo A trasmette esclusivamente sulle celle caratterizzate dalla nuova configurazione ν' ($3j$ e successive). Il nodo B rimane nello stato di double listening fino alla corretta ricezione di un frame $\mathcal{F}$. Se tale frame viene ricevuto in corrispondenza di una cella C_{backup} (`is_main(`j`) = false`), il nodo riconosce che l'altro estremo della comunicazione sta effettivamente utilizzando la nuova configurazione ν'. A questo punto non resta che disabilitare C_{main} ($7i$ e successive), il cui ruolo di cella principale viene assunto da C_{backup} ($7j$ e successive). Come visto per il nodo A, lo *switch* delle celle è effettuato tramite la seguente sequenza di operazioni: `set_configuration(`i`, -)`, `set_configuration(`j`,` ν'`)`, `swap(`C_{main}`,` C_{backup}`)`. Si noti come l'esito della trasmissione del frame di ACK $\mathcal{A}$ in $6j$ non sia di rilevanza ai fini dell'algoritmo. Si è evitata, dunque, la dipendenza ricorsiva descritta in precedenza.

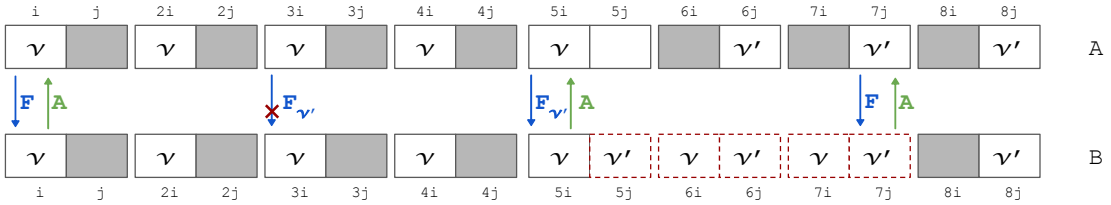
Lo scenario `DATA-NOACK` (Figura 2.5b), invece, è caratterizzato dalla perdita



(a) DATA-ACK



(b) DATA-NOACK



(c) NODATA

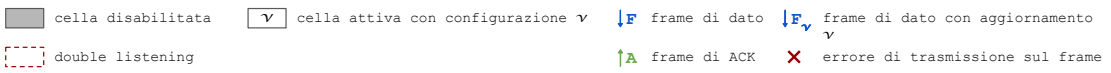


Figura 2.5: Schema protocollo CONSIP

del frame di ACK $\mathcal{A}$ relativo all'aggiornamento della configurazione contenuto in $\mathcal{F}_{\nu'}$ $(3i)$. D'ora in poi, per semplicità, le primitive utilizzate dai nodi per lo *switch* delle celle non vengono riportate esplicitamente. Il nodo A, dunque, non modifica la propria configurazione e procede normalmente con la trasmissione dei pacchetti presenti nella coda. Il nodo B, invece, protrae il proprio stato di double listening fino alla ricezione del frame in $5i$: in questo istante, dopo aver verificato che la comunicazione è avvenuta su una cella C_{main}, B prende atto della mancata conclusione del ν-exchange e disattiva la cella C_{backup}, ritornando alla configurazione iniziale ν. L'utilizzo di CONSIP ha, infatti, garantito la coerenza delle informazioni

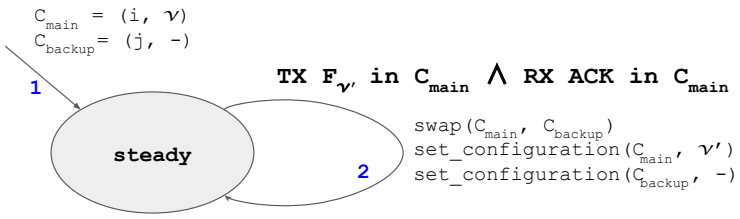
riguardanti ν nel caso di errori di trasmissione. Nell'esempio proposto, tuttavia, il frame $\mathcal{F}_{\nu'}$ ricevuto dal nodo B contiene, a sua volta, un aggiornamento della configurazione. È molto probabile, infatti, che il nodo A ritenti la trasmissione dell'aggiornamento a fronte della mancata ricezione del precedente frame di ACK. Da questo momento ($5i$) in poi, l'algoritmo procede secondo quanto visto nel caso `DATA-ACK`, con l'unico svantaggio di aver posto il nodo B in uno stato di double listening per un periodo di tempo maggiore, con conseguente aumento dei consumi energetici.

Lo scenario `NODATA` (Figura 2.5c), infine, è il più semplice dei tre in quanto il nodo B non riceve mai il frame di aggiornamento ($3i$) e, di conseguenza, il nodo A non può rilevare l'arrivo dell'ACK corrispondente. In questo caso – paragonabile a quanto visto in Figura 2.2 – il nodo trasmettitore può semplicemente decidere di ritentare l'aggiornamento ($5i$).

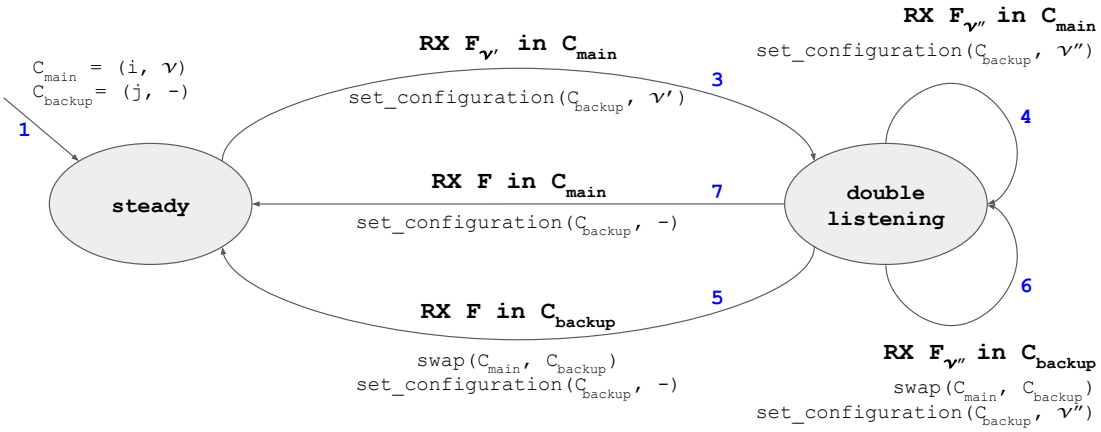
Al fine di fornire una descrizione formale del protocollo, in Figura 2.6 vengono presentate due Finite State Machine (FSM) che illustrano le operazioni effettuate dai due nodi durante il ν-exchange. In entrambi i casi il punto di partenza è lo stato `steady`, durante il quale le trasmissioni avvengono sulla sola cella principale (come espresso dall'arco `{1}`). La configurazione, infatti, prevede che la cella principale trasmetta secondo i criteri espressi da ν, mentre la cella secondaria è temporaneamente disattivata. La notazione utilizzata per la descrizione dello stato di una cella è la seguente: `C role = (pos, configuration)`, dove *role* rappresenta il ruolo – primario o secondario – della cella, mentre la coppia di valori *pos* e *configuration* indica rispettivamente lo slot offset su cui è stata schedulata la cella e il parametro di configurazione utilizzato (`'-'` in caso di cella non attiva).

La FSM relativa al nodo trasmettitore (Figura 2.6a) possiede un solo ulteriore arco (`{2}`), relativo alla trasmissione con conferma del frame di aggiornamento $\mathcal{F}_{\nu'}$. Nel dettaglio, ogni volta che il frame $\mathcal{F}_{\nu'}$ viene trasmesso sulla cella C_{main} e, allo stesso tempo, viene notificata la ricezione dell'ACK corrispondente le azioni che vengono eseguite sono relative allo scambio del ruolo tra le due celle (`swap`) e all'aggiornamento della configurazione delle stesse (`set_configuration`, con parametro ν' per la nuova cella C_{main} e valore nullo `'-'` per la disattivazione della cella C_{backup}). Questa transizione mantiene la FSM sullo stesso stato iniziale: si parla, in questo caso, di arco cappio o *self loop*. Il nodo trasmettitore, in definitiva, opera sempre secondo una configurazione semplice, nella quale le trasmissioni hanno luogo esclusivamente in corrispondenza delle celle C_{main}, mentre le celle di backup restano sempre disabilitate.

Il nodo ricevitore, invece, presenta un numero di archi maggiore in quanto la macchina a stati deve gestire l'ingresso e l'uscita dallo stato di `double listening` (Figura 2.6b). L'arco `{3}` effettua, per l'appunto, la transizione dallo stato iniziale allo stato di double listening, condizionata dalla ricezione, nella cella C_{main}, di un frame $\mathcal{F}_{\nu'}$ contenente l'aggiornamento ν'. Da questo momento in poi, entrambe



(a) FSM nodo trasmettitore



(b) FSM nodo ricevitore

Figura 2.6: Macchina a stati protocollo CONSIP

le celle C_{main} e C_{backup} si pongono periodicamente in ascolto sulla frequenza fisica ricavata rispettivamente dalla configurazione ν e ν'. Di conseguenza, durante questo stato i consumi energetici aumentano in misura proporzionale al numero di celle soggette a double listening. Il ritorno ad uno stato di normalità avviene a seguito del completamento (arco `{5}`) o del fallimento (`{7}`) del ν-exchange. La prima transizione, eseguita in caso di ricezione di un frame $\mathcal{F}$ in corrispondenza di una cella di backup, provoca lo *switch* delle celle, implementato da un'operazione iniziale di `swap` dei ruoli delle stesse e dalla successiva disattivazione della nuova cella C_{backup}. In caso di ricezione di $\mathcal{F}$ su cella principale, invece, l'effetto è semplicemente quello di disattivare la cella C_{backup} e ritornare allo stato iniziale, senza aver effettuato l'aggiornamento. Si noti che, in entrambi i casi, il frame ricevuto non deve contenere alcun aggiornamento della configurazione ν. Nel caso opposto, ossia in presenza di un ulteriore aggiornamento ν'' durante la fase di double listening, i due archi `{4}` e `{6}` continuano a mantenere la FSM sullo stato corrente. Se il frame $\mathcal{F}_{\nu''}$ viene ricevuto su C_{main}, si sovrascrive semplicemente la configurazione relativa alla cella di backup (che passa da ν'a ν''). Se l'aggiornamento è in corrispondenza di C_{backup}, invece, le conseguenze sono due: notificare la corretta terminazione dello scambio

della configurazione ν' e prepararsi a concludere il nuovo scambio, invertendo i ruoli delle celle (`swap`) ed aggiornando la configurazione di C_{backup} (`set_configuration`) al valore espresso da ν''. In generale è possibile che $\nu'' = \nu'$, poiché, in seguito alla perdita del messaggio di ACK, il nodo trasmettitore può decidere di ritentare il ν-exchange relativo alla precedente configurazione ν'.

Nel Capitolo 5 sono riportate una serie di simulazioni effettuate sul protocollo CONSIP, atte a verificare l'efficacia dell'algoritmo dal punto di vista energetico, delle latenze e dell'affidabilità.

Capitolo 3

Probabilistic listing

Nel capitolo precedente si è dimostrato come sia possibile trasmettere in modo coerente un'informazione dal nodo N_{TX} al nodo N_{RX}. Il valore coinvolto nello scambio può essere semanticamente associato a qualsiasi tipo di informazione, anche se l'applicazione principale di CONSIP è la trasmissione di ν', rappresentante la configurazione aggiornata che dev'essere utilizzata per le future comunicazioni tra la coppia di nodi. Lo scambio di informazioni che non alterano le operazioni di selezione del canale logico da utilizzare, infatti, può avvenire attraverso una semplice trasmissione di frame con controllo dell'eventuale ACK corrispondente, come definito nel protocollo TSCH. In questo caso l'*overhead* introdotto da CONSIP – in particolare relativo alla gestione di due canali di comunicazione distinti – non è giustificato.

Il contenuto di ν dipende dallo specifico algoritmo di monitoraggio del canale e di aggiornamento della configurazione utilizzato dai nodi, nonché dalla legge con cui, a partire da ν, si seleziona il canale fisico di trasmissione. I classici algoritmi di whitelisting hanno la peculiarità di definire un sottoinsieme di canali $\mathcal{W} \subseteq \mathcal{C}$ utilizzabili da parte dei nodi, escludendo, di fatto, la restante parte dei canali che ha performato peggio nell'immediato passato. In questo caso, indicando con x il valore corrente di ASN, la x-esima trasmissione può avvenire sul canale $ch_{wh} \in \mathcal{W}$ con probabilità $p_{ch_{wh}} = \frac{1}{|\mathcal{W}|}$ e sul canale $ch_{bl} \notin \mathcal{W}$ con probabilità $p_{ch_{bl}} = 0$. L'esclusione di un canale dalla whitelist è frutto di una decisione, da parte dell'algoritmo, che ha conseguenze nell'immediato futuro: per un certo periodo di tempo, infatti, il canale ch_{bl} non sarà utilizzabile e dovrà essere manualmente reinserito in $\mathcal{W}$, anche solo provvisoriamente, per valutarne nuovamente la qualità. Allo stesso tempo non è possibile associare una scala di priorità ai canali ancora attivi, in quanto le trasmissioni hanno luogo, a lungo termine, con la stessa probabilità su tutti i canali.

Per ovviare a queste limitazioni, in questo capitolo viene proposto un semplice algoritmo di *probabilistic whitelisting*, che associa ad ogni canale logico una determinata probabilità di estrazione. L'algoritmo è composto da tre parti: monitoraggio attivo dello stato dei canali, aggiornamento della configurazione ν e funzione di

estrazione del canale logico.

Come prima operazione è necessario stabilire quale sia il nodo che effettua in modo proattivo l'aggiornamento della configurazione. Negli esempi successivi si prende in considerazione una singola coppia di nodi: il nodo N_{TX}, che genera e trasmette ν', e il nodo N_{RX}, che si limita a ricevere l'aggiornamento e a modificare di conseguenza la propria configurazione. L'algoritmo di monitoraggio attivo dello stato dei canali, operante su N_{TX}, raggruppa i timeslot in intervalli di durata regolare Δt. Durante queste finestre temporali vengono raccolte delle semplici statistiche riguardanti i pacchetti trasmessi da N_{TX} a N_{RX}. Nel dettaglio, il nodo N_{TX} memorizza, per ogni canale ch, il numero di frame trasmessi n_{TX}^{ch} e il numero di ACK ricevuti n_{ACK}^{ch}. Non è necessario, in questo contesto, associare ciascun frame alla ricezione o alla perdita del frame di conferma corrispondente. Quando la finestra temporale Δt si conclude queste informazioni vengono inoltrate all'algoritmo di aggiornamento della configurazione, che ha precedentemente inizializzato i seguenti vettori: $\epsilon = [1.0, ..., 1.0]$ e $p = [\frac{1.0}{N_{ch}}, ..., \frac{1.0}{N_{ch}}]$, entrambi di dimensione N_{ch}. Il primo valore rappresenta una lista di tassi di successo cumulativi delle trasmissioni avvenute sui rispettivi canali, mentre il secondo vettore rappresenta la forma normalizzata di ϵ. Valgono, dunque, le seguenti affermazioni:

$$\epsilon_k \in [0, 1] \quad \forall k \in \{1, ..., N_{ch}\} \tag{3.1}$$

$$\sum_{k=1}^{N_{ch}} p_k = 1 \tag{3.2}$$

Negli esempi successivi si suppone di avere a disposizione $N_{ch} = 16$ canali logici distinti. L'algoritmo effettua il calcolo del tasso di successo dei singoli canali all'interno della finestra temporale appena conclusasi. Il vettore PDR è così calcolato:

$$PDR_k = \begin{cases} \frac{n_{ACK}^k}{n_{TX}^k} & \text{se } n_{TX}^k > 0 \\ \text{invariato} & \text{se } n_{TX}^k = 0 \end{cases} \tag{3.3}$$

Se durante Δt il nodo N_{TX} non ha trasmesso alcun frame in direzione di N_{RX} sul canale logico k ($n_{TX}^k = 0$) il valore di PDR_k rimane invariato e viene ereditato dalla misurazione relativa al Δt precedente, in quanto non è stato possibile misurare direttamente la qualità del link. È possibile che ciò si verifichi se la durata della finestra temporale non è sufficientemente lunga oppure se al canale in questione è associata una probabilità di trasmissione relativamente bassa. Tale vettore rappresenta una stima del livello di qualità della comunicazione durante Δt.

L'algoritmo procede con l'aggiornamento delle probabilità di estrazione dei canali da utilizzare per la finestra temporale successiva. Per fare ciò, il vettore di probabilità p viene modificato in modo da includere l'informazione relativa ai valori di PDR appena calcolati. Questo risultato viene ottenuto per mezzo di due operazioni distinte: l'aggiornamento dei valori cumulativi presenti in ϵ (Equazione (3.4))

e la loro successiva normalizzazione (Equazione (3.5)).

$$\epsilon_k = \alpha \; PDR_k + (1 - \alpha) \; \epsilon_k^{-1} \tag{3.4}$$

$$p_k = \frac{\epsilon_k}{\sum_{i=1}^{N_{ch}} \epsilon_i} \tag{3.5}$$

La funzione di aggiornamento delle probabilità effettua una media pesata tra il valore di PDR, calcolato durante la finestra Δt appena terminata, e il valore contenuto in precedenza in ϵ (l'esponente $^{-1}$ si riferisce al valore calcolato nell'iterazione precedente). In questo modo, in fase di implementazione è possibile regolare il peso relativo all'ultima misurazione di PDR attraverso un'opportuna inizializzazione del parametro α. Nell'implementazione proposta tale valore è costante. È possibile, comunque, introdurre delle ottimizzazioni a Equazione (3.4) introducendo, ad esempio, una dinamicità nel valore di α oppure nella durata degli intervalli Δt. Lo step successivo (Equazione (3.5)) prevede una semplice normalizzazione del vettore appena calcolato, in modo tale da produrre un vettore di probabilità p la cui somma degli elementi è pari a 1.

A questo punto è già possibile terminare l'algoritmo di aggiornamento della configurazione, ponendo $\nu' = p$. Nel seguito vengono illustrati tre step opzionali dell'algoritmo che modificano ulteriormente il vettore di probabilità allo scopo di produrre una lista di valori che possa essere interpretata in modo più diretto dal nodo ricevitore. La prima operazione aggiuntiva prevede la saturazione inferiore dei valori di probabilità. L'algoritmo, per quanto visto finora, assegna una probabilità dinamica di estrazione del singolo canale ch. In caso di interferenza prolungata in corrispondenza della frequenza fisica relativa, la probabilità p_{ch} raggiunge valori prossimi allo 0 dopo un sufficiente lasso di tempo. Nonostante l'effetto della riduzione di p_{ch} sia positivo nell'immediato, in quanto le trasmissioni hanno luogo esclusivamente sui restanti canali, la reintegrazione dello stesso in seguito alla scomparsa dell'interferenza è un procedimento estremamente lungo. Il canale ch viene utilizzato raramente per la trasmissione di frame e, dunque, la velocità di aggiornamento del valore di PDR relativo è limitata (Equazione (3.3)) e, di conseguenza, il valore di probabilità cumulativa ϵ_{ch} tenderà al valore reale dopo un intervallo di tempo sufficientemente lungo e inversamente proporzionale al parametro α (Equazione (3.4)). In caso di $p_{ch} = 0$ è impossibile ripristinare automaticamente il canale in precedenza escluso dalle comunicazioni. L'introduzione di un ulteriore parametro p_{low} consente di limitare inferiormente le probabilità di trasmissione sul canale, a vantaggio di una maggiore reattività dell'algoritmo all'andamento della qualità della trasmissione misurata sul link. È possibile definire un sottoinsieme di canali $\mathcal{T} \subset \mathcal{C}$, di dimensione N_t, contenente tutti i canali scesi sotto soglia e caratterizzati da una probabilità $\dot{p}_t$. La saturazione applicata è espressa dalla formula:

$$p_k = \begin{cases} p_{low} & \text{se } p_k < p_{low} \\ p_k - \frac{\sum_{t=1}^{N_t} (p_{low} - \dot{p}_t)}{N_{ch} - N_t} & \text{altrimenti} \end{cases} \tag{3.6}$$

La configurazione resta invariata se nessun canale è sceso al di sotto della soglia p_{low}.

Tutte le operazioni viste finora necessitano del supporto all'aritmetica *floating point*. Per estendere la compatibilità dell'algoritmo a dispositivi che non dispongono di Floating-Point Unit (FPU) e per ridurre le dimensioni della configurazione ν' da trasmettere al nodo ricevitore, è possibile effettuare un'operazione di quantizzazione delle probabilità calcolate. I valori di $p \in [0, 1]$, caratterizzati da una precisione che dipende dal numero di bit riservati alla mantissa del numero floating point, vengono dunque mappati su una scala di valori numerici interi appartenenti al range $[0, 2^n]$, con n indicante il numero di bit desiderato per rappresentare un singolo valore di probabilità. La precisione di tali valori si riduce alla quantità $step = \frac{1}{2^n}$. Nella pratica ciò non causa problemi poiché, ad esempio, l'utilizzo di $n = 8$ bit comporterebbe una precisione dello 0,4% circa. Gli effetti della quantizzazione, dunque, non sono direttamente osservabili. La quantizzazione è ottenuta per mezzo di:

$$q_k = round(p_k \cdot 2^n) \tag{3.7}$$

Affinché l'algoritmo sia implementabile è necessario che almeno i nodi che effettuano il calcolo di ν' siano dotati di FPU. L'ultimo step aggiuntivo è relativo all'applicazione di una funzione cumulativa al vettore q appena calcolato, in modo da rendere più immediata, da parte di entrambi i nodi, l'estrazione del canale logico su cui comunicare. I valori di q vengono trasformati iterativamente tramite la primitiva `accumulate` ($q = accumulate(q)$) nel seguente modo:

$$q_k = \begin{cases} q_k & \text{se } k = 0 \\ q_k + q_{k-1} & \text{altrimenti} \end{cases} \tag{3.8}$$

Il vettore risultante può essere interpretato come una lista di intervalli, di dimensione variabile, relativi alla probabilità di estrazione del canale identificato dall'indice k.

A conclusione dell'algoritmo, il valore calcolato può essere trasmesso al nodo ricevitore per mezzo di una transazione CONSIP. La configurazione ν', una volta confermata, sarà utilizzata come parametro nel contesto della funzione di estrazione del canale logico, condivisa dai due nodi. La funzione $\mathcal{F}(\nu, \gamma) \rightarrow ch$ estrae il canale ch a partire dalla configurazione attuale e da un generico stato condiviso γ. Nel dettaglio, l'implementazione della funzione $\mathcal{F}$ sui due nodi deve garantire la generazione di un valore pseudo-casuale condiviso $\hat{p}$, utilizzato per l'estrazione dell'indice k, e quindi il canale ch, su cui trasmettere e ricevere frame nella cella schedulata corrente. La coerenza dei valori casuali estratti può essere garantita, ad esempio, per mezzo di opportuni valori memorizzati nello stato condiviso γ. Nel caso in cui il vettore ν' sia stato generato tramite l'intera sequenza di operazioni indicate in precedenza (comprese, dunque, le operazioni opzionali) il canale ch viene ricavato, a partire da $\hat{p}$, tramite il seguente algoritmo:

Input : ν', γ
Output: ch

$\hat{p} = \text{randint}\,(0,\, 2^n, \text{state} = \gamma)$;
for $k,\ p_k \in \nu'$ **do**
  if $\hat{p} < p_k$ **then**
    $ch = k$;
    break;
  end
end
return ch;

Algoritmo 1: Estrazione del canale logico

In Figura 3.1 viene riportata una descrizione dell'algoritmo di aggiornamento della configurazione sotto forma di diagramma a blocchi.

L'utilizzo di una tecnica di whitelisting probabilistico introduce una serie di vantaggi. Il primo è dovuto alla presenza costante di tutti i canali appartenenti a $\mathcal{C}$, seppur con probabilità di estrazione variabile. In questo modo è possibile garantire, allo stesso tempo, la trasmissione della maggior parte dei frame sui canali con un miglior valore misurato di PDR e la possibilità di testare in maniera fluida e continua lo stato dei canali peggiori, in modo da riabilitarli in corrispondenza di misurazioni di qualità del link favorevoli. Se al vettore di probabilità p si applica la saturazione si evitano, inoltre, valori p_k nulli relativi al singolo canale. Se ciò accadesse, tali canali dovrebbero essere necessariamente reintrodotti manualmente facendo affidamento su apposite euristiche per quanto riguarda la scelta delle tempistiche di reinserimento, non riflettendo, così, l'andamento reale della qualità di comunicazione sul link. L'utilizzo di parametri di configurazione variabili, quali α, Δt e n, consente di ottenere un algoritmo altamente configurabile. Il parametro α definisce la reattività dell'algoritmo alle variazioni di PDR misurato. L'utilizzo di valori prossimi allo 0 consente di adottare un approccio conservativo, per cui eventuali variazioni repentine della qualità della comunicazione sul link, in relazione ad uno specifico canale, non provocano modifiche radicali nella configurazione ν'. Al contrario, l'utilizzo di valori prossimo all'1 consente di aggiornare il vettore di probabilità in maniera molto più netta e repentina, con il rischio, tuttavia, di considerare come affidabili degli eventuali picchi temporanei dovuti all'improvvisa comparsa o scomparsa di fonti di interferenza. In modo analogo, la regolazione del parametro Δt consente di definire il grado di dinamicità desiderato. In questo contesto è opportuno definire un valore Δt_{min} inferiore, necessario a garantire la presenza di un numero di frame trasmessi sufficiente a popolare il vettore di PDR misurati. Supponendo di misurare una periodicità di frame trasmessi dal nodo N_{TX} al nodo N_{RX} pari a t_{app} e di avere a disposizione $N_{ch} = |\mathcal{C}|$ canali logici su

cui trasmettere, il valore $\Delta t_{min} = t_{app} \cdot N_{ch}$ consente di valutare, in media, l'esito di una singola trasmissione per ogni canale logico ch. In applicazioni reali potrebbe risultare utile aumentare ulteriormente la durata della finestra Δt in modo da poter disporre di un numero di campioni maggiore per la valutazione dello stato del canale e, incidentalmente, ridurre il consumo energetico introdotto dallo scambio periodico della configurazione aggiornata. A tal proposito, la regolazione del parametro n può incidere sulle prestazioni energetiche dei nodi. La configurazione aggiornata ν', come visto nei capitoli precedenti, viene trasmessa in corrispondenza di un frame generato dall'applicativo e viene inserito all'interno di appositi header. Ridurre la dimensione di tale vettore di probabilità può essere un fattore di rilievo nel calcolo complessivo del consumo energetico, oltre a garantire una minore occupazione dell'header del frame stesso, con conseguente incremento di capacità del payload applicativo.

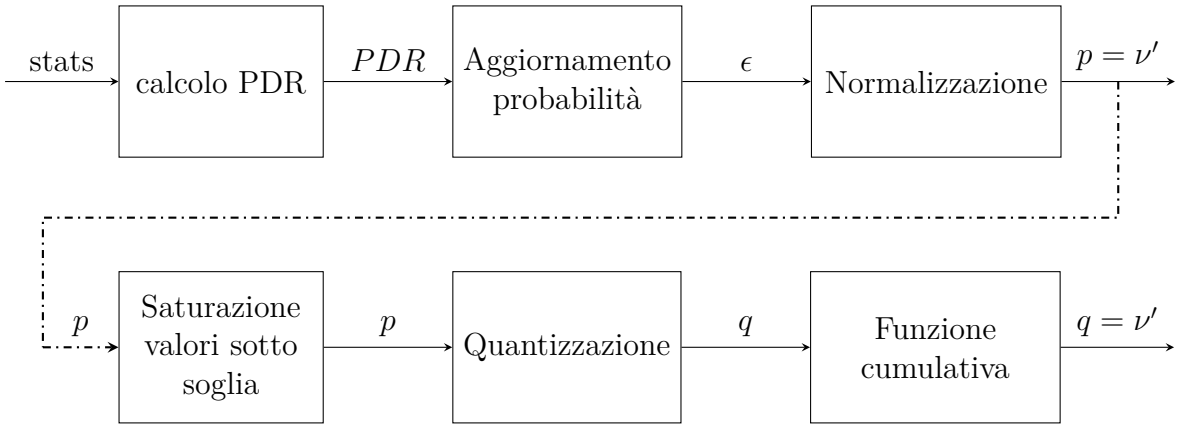


Figura 3.1: Diagramma a blocchi relativo all'algoritmo di aggiornamento della configurazione. I primi tre step sono obbligatori ed è possibile estenderne le funzionalità introducendo dei blocchi aggiuntivi opzionali

Capitolo 4

Implementazione

4.1 OpenWSN

4.1.1 Software di base

Il progetto OpenWSN fornisce un'implementazione open-source dell'intero stack protocollare presentato in Sezione 1.3, rendendo possibile la creazione di reti *mesh* altamente affidabili e a bassissimo consumo energetico su dispositivi embedded reali e disponibili sul mercato [18]. Oltre ad implementare i protocolli necessari per la comunicazione tra tali dispositivi attraverso la rete IPv6 il software fornisce una serie di strumenti necessari al debug, alla gestione della rete tramite interfacce web e a riga di comando e al supporto di funzionalità aggiuntive quali, ad esempio, la capacità dei nodi di rispondere a messaggi Internet Control Message Protocol (ICMP) e la possibilità di effettuare catture di rete utilizzando il software Wireshark[1]. Nonostante siano disponibili progetti alternativi, quali Contiki[2] e TinyOS[3], si è scelto di utilizzare OpenWSN per svariate motivazioni, tra le quali sicuramente la ampia disponibilità di documentazione e risposte a domande frequenti, la possibilità di apportare modifiche al codice sorgente e la compatibilità con i dispositivi a disposizione nell'ambiente di laboratorio. Nelle sperimentazioni sono state utilizzate delle board Industrial Shields OpenMote-B, dotate di microcontrollori Texas Instruments CC2538 ARM-Cortex-M3 a 32 bit e di antenne a 2.4 GHz conformi allo standard IEEE 802.15.4.

OpenWSN è organizzato in due sotto-progetti distinti, ospitati da altrettanti repository GitHub: `openwsn-fw`, contenente il firmware che viene caricato sui nodi, e `openvisualizer`, che consente di interagire con essi attraverso opportune interfacce.

---

[1] Wireshark ®, Wireshark Foundation. Disponibile al link `https://www.wireshark.org/`

[2] `https://github.com/contiki-ng/contiki-ng`

[3] `https://github.com/tp-freeforall/prod`

openwsn-fw

Contiene l'implementazione in linguaggio C dello stack protocollare di riferimento. Il firmware è basato su un modello semplificato di kernel real-time, `openos`, sviluppato dallo stesso gruppo di lavoro di OpenWSN. Tale kernel fornisce uno scheduler di tipo *non-preemptive* ed espone alcune primitive fondamentali, tra cui quelle relative alla creazione, gestione e distruzione di timer interni. Tali timer vengono utilizzati per implementare le operazioni necessarie alla trasmissione e ricezione di frame all'interno delle celle appositamente schedulate, come da Figura 1.6. Tra le cartelle presenti nella struttura principale del repository possiamo identificare le seguenti:

`openstack` racchiude l'implementazione dei livelli *data-link*, rete, trasporto e applicativo (CoAP in particolare)

`openapps` contiene una lista di applicazioni di esempio che possono essere abilitate sui nodi, mettendoli in comunicazione tra loro

`bsp/boards` fornisce le implementazioni dipendenti dall'hardware, associate alle specifiche board, di un insieme di funzioni astratte utilizzate da OpenWSN per eseguire operazioni atomiche sull'hardware quali, ad esempio, la configurazione dei LED o delle antenne. È interessante notare che, in questo contesto, viene fornita anche un'implementazione relativa ad una board simulata in Python. In questo modo è possibile simulare una rete di nodi TSCH anche se non si è in possesso di dispositivi fisici

Dopo aver modificato opportunamente il firmware è possibile programmare le board semplicemente attraverso il comando:

```
sudo scons board=openmote-b-24ghz toolchain=armgcc boardopt=printf
    ↪modules=coap,icmpv6echo,udp apps=cexample,cinfo bootload=/dev/
    ↪ttyUSB1,/dev/ttyUSB5 oos_openwsn
```

Listato 4.1: Caricamento firmware sui dispositivi

openvisualizer

Consente di visualizzare e controllare la rete WSN creata da OpenWSN attraverso la creazione di un'opportuna interfaccia di rete tra la macchina host ed il dispositivo ad essa collegata, che assume il ruolo di *root node* della rete. Come visto in precedenza, la topologia di rete può essere anche associata a nodi simulati. Il software è suddiviso a sua volta in due moduli: `openv-server` e `openv-client`. Il

primo programma, necessario a stabilire la connessione seriale tra la macchina e il nodo root, consente di visualizzare alcune informazioni relative allo stato della rete, stampandole a schermo e producendo uno specifico file di log. È poi possibile invocare appositi comandi `openv-client` per imporre delle azioni ai nodi della rete (`boot` o `root`) oppure per visualizzarne lo stato (`view schedule` o `view neighbors`). Tra le opzioni disponibili c'è anche la possibilità di configurare un web server locale e visualizzare le informazioni attraverso un browser. Un esempio di utilizzo dei due comandi è il seguente (si noti come sia possibile effettuare più richieste parallele nei confronti del singolo server):

```
sudo openv-server --opentun  --
    ↪fw-path "$OPENWSN_PATH" --
    ↪port-mask /dev/ttyUSB*
```

Listato 4.2: Openvisualizer: modulo server

```
1 openv-client root b99b
2 openv-client view schedule ba23
3 openv-client view web
```

Listato 4.3: Openvisualizer: modulo client. Alcuni esempi di comandi

4.1.2 Modifiche apportate

Il codice sorgente dei moduli offerti dal progetto OpenWSN è stato modificato opportunamente allo scopo di effettuare un monitoraggio continuo dello stato della rete. L'ambiente di test consiste in due dispositivi OpenMote-B collocati in un ambiente di laboratorio a circa 3 m di distanza. La stessa sala e quelle circostanti sono soggette ad interferenze esterne di tipo Wi-Fi, Bluetooth ed eventuali altre tecnologie operanti sulla banda 2.4 GHz. Uno dei due nodi, identificato come nodo principale, comunica con una versione leggermente modificata del visualizzatore `openvisualizer` segnalando la ricezione oppure la perdita dell'informazione trasmessa dal secondo nodo. Su quest'ultimo è stata caricata un'applicazione in linguaggio C, appositamente programmata per ottenere i risultati desiderati, denominata `packet-gen`. Tale applicazione pone il dispositivo in una fase di sleep iniziale e, in seguito alla ricezione di un opportuno messaggio CoAP di segnalazione, ha inizio la trasmissione periodica di brevi messaggi, anch'essi trasmessi dal protocollo CoAP, indirizzati verso il nodo principale. Nel payload di tali messaggi sono contenute alcune informazioni che rendono possibile il calcolo della latenza e la rilevazione di messaggi persi, quali l'ASN corrente e un contatore monotonico univoco rappresentato su 16 bit. L'eventuale ricezione di tali messaggi viene segnalata sul visualizzatore, così come l'arrivo del relativo ACK sul nodo che ha trasmesso il frame iniziale. Un esempio di output del file di log, popolato dal visualizzatore `openvisualizer`, è il seguente:

```
1  $ tail -f openv-server.log | grep "packet-gen"
2
3  12:28:45 INFO [packet-gen_root] ASN_high: 0 ASN_middle: 33 ASN_low:
       ↪28891 DATA_RSSI: -58
4  12:28:45 INFO [packet-gen_ack] Counter: 9222 ASN_high: 0 ASN_middle: 33
       ↪ASN_low: 28891 Acked: 1 ACK_RSSI: -55
5  12:28:45 INFO [packet-gen_node] ASN_recv:2191579 Latency:60ms Counter
       ↪:9222
6
7  12:28:45 INFO [packet-gen_root] ASN_high: 0 ASN_middle: 33 ASN_low:
       ↪28894 DATA_RSSI: -72
8  12:28:45 INFO [packet-gen_ack] Counter: 9223 ASN_high: 0 ASN_middle: 33
       ↪ASN_low: 28894 Acked: 1 ACK_RSSI: -70
9  12:28:46 INFO [packet-gen_node] ASN_recv:2191582 Latency:60ms Counter
       ↪:9223
10
11 12:28:46 INFO [packet-gen_ack] Counter: 9224 ASN_high: 0 ASN_middle: 33
       ↪ASN_low: 28897 Acked: 1 ACK_RSSI: -70
12 12:28:46 INFO [packet-gen_root] ASN_high: 0 ASN_middle: 33 ASN_low:
       ↪28897 DATA_RSSI: -55
13 12:28:46 INFO [packet-gen_node] ASN_recv:2191585 Latency:60ms Counter
       ↪:9224
14
15 12:28:46 INFO [packet-gen_ack] Counter: 9225 ASN_high: 0 ASN_middle: 33
       ↪ASN_low: 28900 Acked: 1 ACK_RSSI: -70
16 12:28:46 INFO [packet-gen_root] ASN_high: 0 ASN_middle: 33 ASN_low:
       ↪28900 DATA_RSSI: -73
17 12:28:46 INFO [packet-gen_node] ASN_recv:2191588 Latency:60ms Counter
       ↪:9225
```

Listato 4.4: Campione di pacchetti generati da `packet-gen`. Ciascun gruppo di righe è relativo ad una singola comunicazione e l'ordine relativo delle righe non è deterministico

Come è ben visibile, il file di log può essere raggruppato logicamente in gruppi di tre righe, relative alla singola comunicazione `packet-gen`. Nel dettaglio, il campo `[packet-gen_node]` è stampato dal nodo trasmettitore in corrispondenza della generazione del pacchetto. Se questo viene ricevuto dal nodo ricevitore viene prodotta una riga di `[packet-gen_root]`, nella quale è anche riportato il valore di RSSI corrispondente. L'assenza di tale *entry* determina la mancata ricezione del frame. Per ultimo, la riga `[packet-gen_ack]` è relativa all'eventuale frame di ACK e presenta

anch'essa il valore di RSSI misurato dall'antenna. Il file di log così generato verrà in seguito analizzato da appositi strumenti di *post analysis* (Sezione 4.2). Le modifiche sostanziali apportate al firmware riguardano la creazione dell'applicazione CoAP appena descritta, contenuta nella cartella `openwsn-fw/openapps/packet-gen` e implementata nei seguenti file:

`packet-gen.h` file di intestazione contenente l'inizializzazione della struttura dati utilizzata dall'applicazione. I campi di particolare interesse sono relativi al contatore monotonico crescente (`uint16_t counter`) e alla variabile booleana utilizzata per controllare la generazione di pacchetti (`bool stop`)

`packet-gen.c` contiene il codice applicativo che si occupa di registrare le opportune *callback* nel sistema di gestione degli eventi, di mettersi in ascolto dei messaggi CoAP di inizio e fine di generazione dei pacchetti e di creare, trasmettere e tracciare i log dei frame `packet-gen`. Nel seguente frammento di codice vengono mostrate le operazioni principali:

```
1  void packet_gen_init(void) {
2      packet_gen_vars.stop = TRUE;
3      // register to OpenCoAP module
4      ...
5      packet_gen_vars.desc.callbackRx = &packet_gen_receive;
6      packet_gen_vars.desc.callbackSendDone = &packet_gen_sendDone;
7      coap_register(&packet_gen_vars.desc);
8      ...
9  }
10
11 owerror_t packet_gen_receive(coap_header_iht *coap_header, ...) {
12     switch (coap_header->Code) {
13         case COAP_CODE_REQ_PUT:
14             disableDAO();
15             // send the first packet
16             packet_gen_vars.counter = 0;
17             packet_gen_vars.stop = FALSE;
18             packet_gen_task_cb();
19             break;
20
21         case COAP_CODE_REQ_DELETE:
22             enableDAO();
23             // stop the packet generation
24             packet_gen_task_reset();
25             packet_gen_vars.counter = 0;
```

```
26              packet_gen_vars.stop = TRUE;
27              break;
28  }
29
30  void packet_gen_task_cb(void) {
31      // get a packet
32      pkt = openqueue_getFreePacketBuffer(COMPONENT_packet_gen);
33      // take ownership over that packet
34      pkt->creator = COMPONENT_packet_gen;
35      pkt->owner = COMPONENT_packet_gen;
36
37      // add payload
38      if (packetfunctions_reserveHeader(&pkt, 15) == E_FAIL) {
39          openqueue_freePacketBuffer(pkt);
40          return;
41      }
42
43      // add counter
44      uint8_t len = 0;
45      pkt->payload[len++] = (uint8_t)(packet_gen_vars.counter & 0x00ff
            ↪);
46      pkt->payload[len++] = (uint8_t)((packet_gen_vars.counter & 0
            ↪xff00) >> 8);
47      // add asn
48      ieee154e_getAsn(asnArray);
49      memcpy(&pkt->payload[len], asnArray, sizeof(asnArray));
50      len += sizeof(asnArray);
51      ...
52      // send
53      outcome = coap_send(pkt, COAP_CODE_REQ_PUT, ...);
54  }
55
56  void packet_gen_sendDone(OpenQueueEntry_t *msg, owerror_t error) {
57      uint8_t asnArray[5];
58
59      ieee154e_getAsn(asnArray);
60      acked = error == E_SUCCESS;
61      ack_rssi = acked ? msg->ack_packet->l1_rssi : 0;
62
63      openserial_printf( "[packet-gen_node] Counter: %d ASN_high: %d
            ↪ASN_middle: %d ASN_low: %d Acked: %d ACK_RSSI: -%d\n",
64          (uint16_t) packet_gen_vars.counter,
```

```
65          (uint16_t) asnArray[4],
66          (uint16_t) (asnArray[2] + (asnArray[3] << 8)),
67          (uint16_t) (asnArray[0] + (asnArray[1] << 8)),
68          (uint16_t) acked,
69          (uint16_t) ack_rssi );
70
71      // restart when counter reaches 16-bit max value.
72      if (packet_gen_vars.counter == 0xFFFF) {
73          packet_gen_task_reset();
74          packet_gen_task_start();
75      }   else {
76          packet_gen_vars.counter++;
77      }
78
79      // send the next packet
80      packet_gen_task_cb();
81  }
```

Listato 4.5: Operazioni di inizializzazione e trasmissione di frame `packet-gen`

Durante la trasmissione dei frame la generazione di messaggi DAO viene disattivata (`riga 14`), per poi essere riabilitata quando la trasmissione dei frame viene interrotta (`riga 22`), allo scopo di mantenere una costanza nella periodicità di creazione e trasmissione dei frame `packet-gen`. In assenza di questa operazione di disattivazione i messaggi di DAO, generati periodicamente secondo un apposito parametro di configurazione (Tabella 4.1), andrebbero a sovrapporsi ai sopracitati messaggi all'interno delle code di trasmissione. Le ultime istruzioni della funzione `packet_gen_sendDone` consentono di rilevare automaticamente il raggiungimento del valore massimo rappresentabile dal campo contatore e di ricominciare con una nuova generazione di pacchetti. A seguito della trasmissione del pacchetto precedente, il successivo viene subito schedulato ed inserito nella coda di trasmissione. In questo modo non si introducono ritardi dovuti alla gestione di timer interni e, dunque, la periodicità delle trasmissioni successive di frame è costante e dipende dal numero e dalla posizione delle celle schedulate tra la coppia di nodi in questione

`packet-gen.py` Fornisce un'interfaccia nei confronti dell'applicativo presente sui nodi. In particolare è possibile iniziare una nuova generazione di pacchetti (`python packet-gen.py start`) oppure terminarne una già in corso (`python packet-gen.py stop`)

Affinché la generazione di pacchetti sia effettivamente attiva è necessario aggiungere l'applicazione alla lista di `apps` in fase di caricamento del firmware sui nodi

(Listato 4.1). Il secondo set di modifiche riguarda la configurazione dei parametri e delle costanti relative alla costruzione della rete TSCH in OpenWSN.

Parametro	Valore originale	Valore aggiornato
SLOTDURATION	20 ms	
SLOTFRAME_LENGTH	101	3
TX_RETRIES	15	0
TX_POWER	0 dBm	
NUM_CHANNELS	16	
CH_TEMPLATE	[5, 6, 12, 7, 15, 4, 14, 11, 8, 0, 1, 2, 13, 3, 9, 10]	
DIO_PERIOD	10 s	
DAO_PERIOD	60 s	DAO disabilitati durante la `packet-gen`

Tabella 4.1: Parametri di configurazione di OpenWSN. Laddove non espressamente indicato il valore utilizzato coincide con l'originale

In particolare, la durata dello slotframe è stata ridotta a tre timeslot, associati alla cella condivisa di controllo e alle due celle necessarie per la comunicazione bidirezionale. La maggior parte del traffico, comunque, fluisce dal nodo secondario al nodo principale. In questo modo si ottengono una serie di comportamenti desiderati. In primo luogo la periodicità dei messaggi di controllo si riduce notevolmente, garantendo una maggiore rapidità nella costruzione della topologia ed un incremento nella stabilità della rete. In secondo luogo la coda di trasmissione del nodo secondario che comunica con il nodo principale può essere svuotata molto più rapidamente e con latenze costanti, continuando a garantire la completa esplorazione del vettore di canali H nell'arco di 16 trasmissioni successive. Questa proprietà è soddisfatta dall'utilizzo di valori di channel offset costanti e di valori di $N_{ch} = 16$ e $N_{slots} = 3$ primi tra loro. La seconda modifica principale riguarda il parametro TX_RETRIES che, assumendo il valore nullo, previene tentativi successivi di trasmissione del frame in seguito alla mancata ricezione dell'ACK corrispondente. In Tabella 4.1 vengono riportati i parametri di configurazione principali. La generazione di pacchetti secondo la configurazione riportata consente di testare lo stato della rete ogni SLOTFRAME_LENGTH$\cdot$SLOTDURATION $= 60$ ms e l'esito della trasmissione su tutti i canali ogni 60 ms $\cdot$ NUM_CHANNELS $= 960$ ms.

4.2 Aggregazione e analisi dei dati raccolti

Il file di log popolato dallo scambio periodico di messaggi `packet-gen` può essere punto di partenza per approfondite operazioni di *post analysis*. Lo scopo è quello di ottenere una panoramica dello stato della rete durante una finestra temporale di lunghezza stabilita. Le statistiche risultanti dall'analisi dei dati forniscono sia il livello medio finale della qualità della trasmissione sia l'evoluzione temporale della stessa. Nel seguito si analizzano gli strumenti utilizzati per l'analisi e l'aggregazione dei dati in ingresso.

```
$ python3 framestats.py --help

Usage: framestats.py [OPTIONS] COMMAND [ARGS]...
Commands:
  store      Parse the logfile and store packets info to the databases
  frequency  Print tabular information about frames and frequencies
  output     Print the packets information stored in the databases
  window     Produce rolling window for FDP in order to study time
     ↪evolution
```

Listato 4.6: `framestats.py`: comandi disponibili

Il modulo principale, denominato `framestats.py`, espone una serie di comandi visualizzabili attraverso il comando di `help` (Listato 4.6). In particolare, il comando di `store` è considerato l'*entry point* per le successive analisi poiché ha il compito di analizzare il file di log fornito e di produrre una serie di database testuali relativi alle singole sessioni di generazione di pacchetti `packet-gen`. I database prodotti, i cui *filename* seguono lo schema `gen_00001.txt, ..., gen_0000N.txt`, contengono le informazioni aggregate secondo il modello di rappresentazione Comma Separated Value (CSV). In Tabella 4.2 vengono riportare le prime righe estratte da un database associato ad una misurazione realmente effettuata in laboratorio. A partire da questi dati è possibile estrarre le informazioni riguardanti l'andamento medio complessivo globale e relativo ad ogni frequenza. Il comando `framestats.py frequency`, applicato ad un determinato numero di database, genera una tabella riassuntiva nella quale vengono riportati, per ogni canale e globalmente per l'intera rete, i valori medi di Frame Delivery Probability (FDP), ACK Delivery Probability (ACKDP), RSSI e latenza media. Il comando `framestats.py window`, invece, riceve in input un database testuale e restituisce, per ogni frequenza fisica utilizzata, un file rappresentante l'andamento temporale dello stato della rete sotto forma di finestra mobile (*rolling window*). Tali finestre mobili sono configurabili per mezzo dei due parametri `size` e `step`, associati rispettivamente al numero di campioni

counter	asn	received	latency	frequency	acked	rssi	ack_rssi
0	1297	1	60	13	1	-60	-54.0
1	1300	1	60	20	1	-58	-55.0
2	1303	1	60	17	1	-55	-55.0
3	1306	1	60	26	0	-59	0.0
4	1309	1	60	22	1	-61	-57.0
5	1312	1	60	12	1	-53	-51.0
6	1315	1	60	14	1	-62	-51.0
7	1318	0	0	16	0	0.0	0.0
8	1321	1	60	18	1	-62	-51.0

Tabella 4.2: Esempio di database generato dal comando `framestats.py store`. Nei dati proposti si possono osservare i tre scenari: perdita di pacchetto (indice 7), perdita di ACK (indice 3) e trasmissione conclusa con successo (restanti indici)

presi in considerazione per il calcolo della media in ogni intervallo temporale e allo spostamento effettuato in vista del calcolo successivo. In aggiunta ai file relativi alle singole frequenze viene generato anche un file globale. Il contenuto di questi file è di difficile interpretazione diretta e, dunque, l'ultima operazione di *post analysis* consiste nel tracciare dei grafici a linee che consentono di visualizzare l'andamento temporale dei singoli canali. Nel Capitolo 5 vengono presentati alcuni esempi di output dei comandi sopracitati così come alcuni grafici significativi.

4.3 TSCHpredictor

4.3.1 Software di base

In modo parallelo rispetto alla raccolta di statistiche riguardanti lo stato di uno scenario di rete reale sono state svolte un cospicuo numero di simulazioni atte a validare l'efficacia e la stabilità degli algoritmi presentati in precedenza. A tale scopo è stato utilizzato il simulatore ad eventi discreti TSCHpredictor, presentato in [26] e rivisto in [27]. Tale simulatore, sviluppato in Python per mezzo del framework `SimPy`, permette di implementare e testare in maniera rapida nuovi algoritmi, focalizzandosi sull'analisi dei soli fattori di interesse quali le latenze, l'affidabilità e i consumi energetici. A differenza di altri simulatori di reti TSCH disponibili, quali

6TiSCH simulator [40] e TSCH-Sim [41], infatti, in questo simulatore viene implementato solo un sottoinsieme delle funzionalità offerte dallo standard TSCH e dagli altri protocolli appartenenti allo stack. In particolare, le simulazioni vengono pilotate da un apposito file di configurazione che, tra le altre cose, consente di controllare i seguenti aspetti della rete TSCH simulata:

- la durata della simulazione, espressa in numero di timeslot
- il modello di canale utilizzato, esprimibile in termini di probabilità di errore sul pacchetto (FDP) e sul frame di conferma (ACKDP). È possibile definire valori differenti per ogni cella schedulata nella matrice TSCH
- uno o più flussi di pacchetti periodici che transitano dal nodo trasmettitore a quello ricevitore, eventualmente attraversando una lista di *hop*. In questa fase è necessario definire il routing associato ai rispettivi flussi, poiché il simulatore non implementa algoritmi di creazione dinamica della topologia della rete, offrendo una maggiore predicibilità dei risultati. I flussi al momento supportati sono di tipo periodico e, di conseguenza, è necessario esplicitare l'intervallo di generazione tra pacchetti successivi
- il modello energetico associato ai nodi simulati e necessario per effettuare il calcolo finale dei consumi energetici. Nella versione base del simulatore viene utilizzato il modello energetico contenuto in [26] – e derivato da [42] – che indica la quantità di energia necessaria a trasmettere e ricevere un frame di dato o di ACK. L'output finale riporta l'energia totale consumata da ogni nodo in termini assoluti, espressa in J, così come la quantità di potenza consumata, espressa in µW

Si noti come la caratteristica di periodicità dei flussi simulabili in TSCHpredictor non rappresenta un vero ostacolo sia perché la maggior parte del traffico dati in reti TSCH reali è effettivamente di tipo periodico sia perché è possibile modellare del traffico più complesso sovrapponendo più flussi paralleli con periodicità differenti (eventualmente multiple tra loro).

Il punto di ingresso del simulatore è il file `TSCHpredictor.py`, che inizializza il generatore di eventi discreti `SimPy` configurando gli eventi periodici relativi alle celle schedulate della matrice TSCH e gli eventi relativi alla generazione periodica dei pacchetti. A fine simulazione, inoltre, vengono stampati a schermo i risultati relativi al consumo energetico dei nodi. Sono presenti ulteriori file che implementano le varie funzionalità supportate dal simulatore, che vengono di seguito brevemente descritte ed associate alle rispettive classi Python di appartenenza:

`TSCH` è il fulcro del simulatore. Si occupa inizialmente di leggere il file di configurazione relativo alla matrice TSCH e di schedulare le celle in esso contenute. Fornisce, poi, l'implementazione della funzione periodica, registrata nel generatore di eventi discreti da parte del modulo principale, che gestisce le

operazioni di ricezione e trasmissione dei frame in corrispondenza delle celle schedulate. In breve, tale funzione estrae il primo pacchetto presente nella coda di trasmissione relativa alla coppia di nodi coinvolti nella cella corrente e, se la coda non è vuota, ha luogo il tentativo di trasmissione del pacchetto estratto, che avrà una probabilità di errore sul frame di dato ϵ_f e sul frame di conferma ϵ_{ACK}. Per ogni evento il modulo aggiorna delle opportune statistiche sul numero di pacchetti trasmessi, ricevuti e persi necessarie al computo energetico finale. In questo contesto le operazioni critiche sono quelle associate al riconoscimento di pacchetti duplicati lato ricevitore e causati da una mancata ricezione del frame di conferma da parte del nodo trasmettitore. Tali frame, ovviamente, non vengono inoltrati a destinazione da parte del nodo intermedio che li riceve, in quanto ne è stata già trasmessa la prima copia.

`Packet` modella l'entità relativa al frame di dato che viene inoltrato, di nodo in nodo, dal trasmettitore al destinatario della trasmissione. Incapsula al suo interno tutte le informazioni necessarie alla rilevazione di duplicati e al calcolo dell'energia consumata

`Node` fornisce l'implementazione associata ai singoli nodi della topologia di rete espressa dal file di configurazione. Tale classe è incaricata di controllare i diversi link di comunicazione con gli altri nodi della rete, di gestire le code di trasmissione ad esse associate e di fornire gli strumenti necessari per il resoconto energetico finale

`Generator` rende disponibile al modulo principale la funzione periodica che si occupa della creazione di un nuovo pacchetto da inserire nella coda di trasmissione del nodo sorgente. È possibile, in fase di creazione del pacchetto, inserire eventuali campi aggiuntivi all'interno dell'IE del pacchetto stesso

`Logger` offre una semplice interfaccia che consente di tenere traccia dei pacchetti generati e, al termine della simulazione, di produrre un file di log contenente tale lista

4.3.2 Modifiche apportate

Il software TSCHpredictor è stato oggetto di modifiche orientate ad implementare gli algoritmi di scambio coerente e di whitelisting probabilistico. Affinché i due algoritmi potessero essere effettivamente implementati sul simulatore sono state apportate anche delle modifiche preliminari alla struttura del codice e alla modalità di definizione del modello del canale.

Modello di canale

Le modalità di modellazione dello stato del canale sono state espanse, dato che la versione base del software fornisce solamente la possibilità di associare valori costanti di FDP e ACKDP per ogni cella schedulata, a prescindere dal canale logico di trasmissione. Le modifiche introdotte consentono di modellare lo stato del canale in diverse modalità: globale, fissa per ogni canale, variabile, basata su un file di campioni acquisiti dal monitoraggio di una rete TSCH reale. Sono possibili, inoltre, tecniche miste. Il file di configurazione è formattato secondo lo standard YAML. In Listato 4.7 e Listato 4.8 sono riportati due esempi di configurazione, rispettivamente associati alla modellazione manuale dei canali e all'utilizzo di campioni precedentemente raccolti.

```
1  cell 1:
2    slot_offset: 1
3    channel_offset: 1
4    source: 1
5    destination: 0
6    fdp:
7      - 1: 0.981136
8      - 2: 0.960402
9      - 3: 0.95939
10     - 4: 0.978393
11     - 5, 6, 7, 8: [ 10000 0.92, 11000 0.65 ]
12     - other: 0.988644
13   ackdp: 0.97
```

Listato 4.7: Modello di canale variabile

Nel caso di Listato 4.7 si è deciso di associare una probabilità di successo della trasmissione del frame di conferma fissa ed indipendente dal canale logico utilizzato. La probabilità di successo della trasmissione del frame di dato, invece, può essere costante nel tempo per determinati canali (ad esempio `riga 7`) o per tutti quelli rimanenti (`riga 12`), oppure può seguire un andamento variabile secondo la forma d'onda passata come chiave (`riga 11`). In questo caso occorre indicare delle coppie di valori `(time, value)` relative agli istanti temporali di variazione della probabilità. Nel caso di modellazione del canale per mezzo di campioni (Listato 4.8), il formato del file passato come parametro è il CSV e ogni riga deve contenere i seguenti campi: `ASN, frequency, received, acked`. Il formato di tale file riproduce fedelmente il database di informazioni che può essere estratto dall'analisi del traffico che fluisce tra dispositivi reali. Lo scopo di tale configurazione, infatti, è quello di utilizzare

per le simulazioni un modello di canale derivato da condizioni reali.

```
cell 1:
  slot_offset: 1
  channel_offset: 1
  source: 1
  destination: 0
  path: "../packet-gen.txt"
```

Listato 4.8: Modello di canale associato a file di campioni

La modellazione della configurazione desiderata viene implementata dalla classe `Channel`. In tale porzione di codice vengono forniti i mezzi per recuperare le probabilità FDP e ACKDP a partire dal canale logico e dal valore di ASN di riferimento. Il modello di canale così implementato è racchiuso, insieme ad altri parametri, nella classe `Cell`, implementata per estendere le funzionalità associate alle celle schedulate in precedenza contenute direttamente nel file `TSCH.py`. Il contenuto di questa classe viene approfondito nella sezione successiva, relativa al protocollo CONSIP.

Scambio coerente di informazione

Il primo algoritmo implementato sul simulatore è quello descritto nell'ambito del protocollo di scambio coerente CONSIP (Capitolo 2). Le modifiche sostanziali riguardano l'introduzione, per ogni cella presente nel file di configurazione della matrice TSCH, di una cella di backup, inizialmente impostata su uno stato di inattività. Le operazioni di creazione e accoppiamento delle due celle sono descritte nel Listato 4.9. È possibile visualizzare le *policy* applicate alla selezione del channel offset (invariato) e dello slot offset (cioè successivo, equispaziato o espresso esplicitamente) della cella secondaria. Le righe `32` e `33` mettono in relazione le due celle per garantire le operazioni di *switch* e di disattivazione ed attivazione delle stesse.

```
class TSCH:
  def __init__(self, filename):
    self.matrix = []

    # Lettura della matrice TSCH
    with open(filename) as f:
      config = yaml.safe_load(f)
      for cell in config.values():
        ...
```

```
10          # Crea una nuova cella principale per ogni entry nel file di
                ↪configurazione
11          main_cell = Cell(slot_offset, channel_offset, source, dest, fdp,
                ↪ ackdp)
12          self.matrix.append(main_cell)
13
14          # Se il whitelisting e' abilitato, crea la cella di backup
                ↪corrispondente
15          if whitelisting:
16            slots = global_config.config.get("N_slots")
17            policy = global_config.config.get("backup_cell_policy", "
                  ↪spaced").lower()
18
19            # Slot offset policy: indicata nel file di configurazione
20            if policy == "spaced":
21              slot_offset = (slot_offset + slots // 2) % slots
22            elif policy == "next":
23              slot_offset = (slot_offset + 1) % slots
24            elif policy.startswith("fixed"):
25              slot_offset = int(policy.split()[1])
26
27            # Channel offset policy: utilizza lo stesso channel offset
                  ↪della cella principale
28            backup_cell = Cell(slot_offset, channel_offset, source, dest,
                  ↪fdp, ackdp, active_tx=False, active_rx=False)
29            self.matrix.append(backup_cell)
30
31            # Crea associazione tra le due celle
32            main_cell.set_other(backup_cell)
33            backup_cell.set_other(main_cell)
```

Listato 4.9: Creazione cella principale e di backup

Nello stesso file `TSCH.py` è stata modificata anche la funzione `tsch_event()`. Un estratto di tale funzione, al netto delle operazioni di controllo ed aggiornamento delle statistiche relative al calcolo dei consumi energetici, è presentato in Listato 4.10. In primo luogo si valuta lo stato di attività della cella corrente, principale o secondaria che sia, e si verifica l'eventuale presenza di pacchetti nella coda di trasmissione associata al nodo ricevitore. Si noti che il nodo ricevitore non corrisponde necessariamente con il destinatario del pacchetto, in quanto potrebbe semplicemente agire da nodo intermediario. Nel caso in cui sia presente almeno un pacchetto da inoltrare i due nodi vengono interrogati sul canale

logico da utilizzare per la trasmissione sull'ASN corrente per mezzo della chiamata alla funzione `cell.get_frequency(asn)`. Il canale restituito viene utilizzato come argomento della funzione `cell.get_fdp(frequency, asn)`, che restituisce la probabilità di successo per la trasmissione del frame di dato. In modo analogo `cell.get_ackdp(frequency, asn)` indica la probabilità di successo relativa al frame di ACK corrispondente. A questo punto entrambi i nodi generano lo stesso numero pseudo-casuale `p'` e la trasmissione del frame di dato ha luogo se `p' < fdp`, mentre fallisce in caso contrario. Analogamente, l'ACK viene trasmesso solo se il secondo numero pseudo-casuale `p" < ackdp` e, ovviamente, se la precedente trasmissione del frame di dato ha avuto successo.

Le modifiche principali apportate alla funzione riguardano la gestione delle celle di backup e la rilevazione di informazione ν', qui denominata `array`, presente all'interno dell'IE del pacchetto da trasmettere. Il nodo trasmettitore, in caso di conferma del frame contenente la nuova configurazione, notifica la conclusione dello scambio CONSIP attraverso la funzione `cell.switch(array)`. Il nodo ricevitore, invece, è in grado di abilitare la cella secondaria in caso di presenza della configurazione `array` nell'IE (`cell.add(array)`) e di disattivarla in caso di ricezione di un frame valido in corrispondenza di una cella attiva schedulata durante la fase di double listening (`cell.remove()`).

```
1  def tsch_event(env, cell, conf, nodes):
2
3    # Esecuzione periodica della routine di trasmissione e ricezione dei
       ↪frame
4    while True:
5      asn = env.now
6
7      src_node = nodes.get(cell.source)
8      pkt = src_node.in_queue(cell.dest)
9
10     # Se il nodo e' attivo in trasmissione ed e' presente un pacchetto
         ↪da trasmettere al nodo ricevitore associato alla cella
         ↪corrente
11     if (
12       (cell.active_tx if whitelisting else True)
13       and pkt is not None
14       and pkt.get_next_node(src_node) == cell.dest
15     ):
16       ...
17       # Ottieni frequenza fisica di trasmissione
18       frequency = cell.get_frequency()
```

```
19
20        # Estrai l'eventuale informazione aggiuntiva associata al
              ↪protocollo CONSIP
21        array = pkt.ie.get("array")
22
23        # Simula trasmissione del frame di dato
24        fdp, ackdp = cell.get_probabilities(frequency, asn)
25        if random.uniform(0, 1) < fdp:
26          # Il frame di dato ha raggiunto il nodo ricevitore
27          if whitelisting and cell.is_double_listening() and not array:
28            # CONSIP, lato ricevitore: disattivazione della cella opposta.
                  ↪ Fine del periodo di double listening. Mantenere il
                  ↪double listening in caso di sovrascrittura dell'array di
                  ↪ configurazione. Da ora in poi le trasmissioni
                  ↪avverranno secondo la configurazione aggiornata
29            cell.remove()
30          elif whitelisting and array:
31            # CONSIP, lato ricevitore: abilitazione della cella secondaria
                  ↪. Si entra nel double listening mode
32            cell.add(array)
33
34          # Simula trasmissione del frame di ACK (condizionata dalla
                ↪precedente trasmissione)
35          if random.uniform(0, 1) < ackdp:
36            # Il frame di ACK ha raggiunto il nodo trasmettitore
37            if whitelisting and array:
38              # CONSIP, lato trasmettitore: effettua lo switch delle celle
                    ↪ e termina l'algoritmo. Da ora in poi le trasmissioni
                    ↪avverranno secondo la configurazione aggiornata
39              cell.switch(array)
40            ...
41
42          else:
43            # Il frame di ACK *non* ha raggiunto il nodo trasmettitore. Il
                  ↪ pacchetto verra' ritrasmesso in futuro e verra'
                  ↪considerato come duplicato
44            ...
45
46          # In ogni caso il pacchetto e' giunto a destinazione. Vengono
                ↪eseguiti gli aggiornamenti delle statistiche e l'inoltro
                ↪eventuale del pacchetto
47          ...
```

```
48
49        else:
50          # Il frame di dato *non* ha raggiunto il nodo ricevitore.
              ↪Schedulazione del tentativo successivo di ritrasmissione e
              ↪, se raggiunto il limite, notifica della perdita del
              ↪pacchetto
51          ...
52
53      else:
54        # Protocollo CONSIP: il nodo in questione e' soggetto a double
            ↪listening e non e' schedulata alcuna trasmissione. Il
            ↪consumo energetico e' comunque incrementato
55        ...
56
57      # Rischedulazione della funzione in base alla periodicita' della
          ↪cella
58      yield env.timeout(conf["N_slots"])
```

Listato 4.10: Implementazione del protocollo CONSIP sul simulatore

La classe `Cell` modella le celle di comunicazione schedulate nella matrice TSCH e contiene al suo interno tutti gli attributi rappresentati nel file di configurazione, oltre al riferimento alla cella speculare nel caso in cui l'opzione relativa al whitelisting sia stata abilitata. Viene qui memorizzata, inoltre, la *hopping sequence* correntemente in uso. Il metodo `get_frequency()`, sulla base di tale informazione, restituisce la frequenza fisica da utilizzare per la trasmissione.

Nella classe `Node`, invece, sono state aggiunte delle porzioni di codice per gestire la generazione periodica di nuovi `array` e per estendere il modello energetico utilizzato dalle simulazioni. Il metodo `queue()`, già presente nella versione base e incaricato di inserire nuovi pacchetti nella coda di trasmissione, è stato ampliato con lo scopo di creare un `LinkMonitor` (si veda la sezione successiva) per il monitoraggio del traffico dati e di calcolare ed inserire nel pacchetto la configurazione di canali ν' aggiornata.

Il modello energetico utilizzato per le simulazioni, i cui risultati sono presentati nel capitolo successivo, è descritto in [43]. In particolare, l'energia associata alla trasmissione di un frame di dato è $E_{tx} = E_{tx0} + e_{tx} \cdot L_{tot}$, mentre l'energia consumata per ricevere un frame di dato è pari a $E_{rx} = E_{rx0} + e_{rx} \cdot L_{tot}$, con L_{tot} pari alla dimensione in B del pacchetto. Gli altri parametri vengono esplicitati nella sezione relativa al file di configurazione utilizzato nelle simulazioni. I consumi energetici indipendenti dalla dimensione del pacchetto, invece, sono relativi alla trasmissione e ricezione di un frame di ACK, rispettivamente $E_{tx}^{ACK} = 106\,\mu J$ e $E_{rx}^{ACK} = 79\,\mu J$, e al periodo di idle listening, con $E_{listen} = 138\,\mu J$.

Whitelisting probabilistico

Le tecniche di whitelisting probabilistico presentate in Capitolo 3 sono state implementate prevalentemente nelle classi `LinkMonitor` e `Node`. Il modello sviluppato coincide con la versione completa dell'algoritmo raffigurata in Figura 3.1. La classe `Node` è incaricata di istanziare un `monitor` per ogni nodo con cui comunica, a prescindere dal numero di celle schedulate per la comunicazione tra i due estremi. L'algoritmo di monitoraggio attivo dello stato dei canali è implementato semplicemente dal metodo `tx(frequency, acked, ASN)`, proprio di ogni monitor. Tale funzione aggiorna le statistiche riguardanti la trasmissione appena conclusa sulla frequenza fisica e in corrispondenza dell'ASN specificati. Il parametro booleano `acked` fa riferimento alla mancata o avvenuta conclusione dello scambio di informazione. La classe `LinkMonitor` espone, inoltre, i seguenti metodi:

`get_array` restituisce la configurazione di canali generata e non ancora confermata

`reset` azzera tale configurazione in seguito alla corretta conclusione dello scambio CONSIP

`trigger_update` si occupa di generare la nuova configurazione a partire dalle statistiche raccolte nella finestra temporale appena terminata. Si tratta di una funzione periodica, la cui periodicità è configurabile nel file di log. Nel Listato 4.11 sono riportate le istruzioni eseguite allo scopo di aggiornare la configurazione.

```
1  def _update_probability(self, pdr: List[float]) -> Tuple[List[float
       ↪], List[float]]:
2    # Aggiornamento del vettore di probabilita' cumulativo attraverso
         ↪ l'utilizzo del parametro alpha
3    epsilon = [
4      self.ALPHA * pdr[i] + (1 - self.ALPHA) * e
5      for i, e in enumerate(self.epsilon)
6    ]
7
8    # Normalizzazione
9    probability = [e / sum(epsilon) for e in epsilon]
10
11   # Saturazione inferiore
12   if under_threshold := [
13     self.THRESHOLD_LOW - p for p in probability if p < self.
         ↪THRESHOLD_LOW
```

```
14    ]:
15      uniform_decrement = sum(under_threshold) / (
16        len(probability) - len(under_threshold)
17      )
18      probability = [
19        self.THRESHOLD_LOW if p < self.THRESHOLD_LOW else p -
              ↪uniform_decrement
20        for p in probability
21      ]
22
23    return probability, epsilon
24
25
26  def trigger_update(self, env: Environment):
27    """Aggiornamento periodico della configurazione"""
28
29    yield env.timeout(self.PERIOD)
30
31    while True:
32      asn = env.now
33
34      # Calcolo del vettore PDR. Il valore rimane invariato in caso
            ↪di assenza di campioni per il canale di riferimento
35      pdr = [mean(f) if f else self.epsilon[i] for i, f in enumerate(
            ↪self.packet_stats)]
36
37      # Metodo privato che aggiorna il vettore 'epsilon' e la sua
            ↪forma normalizzata 'probability'
38      self.probability, self.epsilon = self._update_probability(pdr)
39
40      # Quantizzazione del vettore. Da ora in poi si opera in
            ↪aritmetica intera
41      quantization = [int(p // self.step) for p in self.probability]
42
43      # Funzione cumulativa
44      self.new_array = list(accumulate(quantization))
45
46      # Preparazione per la misurazione successiva
47      self.packet_stats = [[] for _ in range(16)]
48
49      yield env.timeout(self.PERIOD)
```

Listato 4.11: Aggiornamento della configurazione `array`

File di configurazione

Il file di configurazione è stato esteso rispetto alla versione originale, in modo da consentire la personalizzazione di un numero maggiore di parametri. In Tabella 4.3 vengono riportati il nome, il dominio e alcune note aggiuntive riferite ai parametri supportati. I parametri energetici utilizzati, invece, sono i seguenti: $E_{tx0} = 7\,\mu J$, $E_{rx0} = 65\,\mu J$, $e_{tx} = 2\,\mu J/B$ e $e_{rx} = 1.3\,\mu J/B$.

Parametro	Tipo	Dominio	Note
sim_duration	`int`	$[0, \infty)$	Numero di timeslot
n_slots	`int`	$[1, \infty)$	Dimensione dello slotframe
max_tries	`int`	$[1, \infty)$	Numero di tentativi di trasmissione, incluso il primo
TSCH_matrix	`string`		Path relativo al file di configurazione della matrice
res	`string`		Path relativo al file di log
power_model	`string`		DOI relativo all'articolo scientifico in cui viene presentato il modello energetico utilizzato
verbose	`bool`		Esprime il livello di verbosità dell'output desiderato
whitelisting	`bool`		Abilita la caratteristica di whitelisting. Requisito dei parametri successivi
backup_policy	`string`	{"spaced", "next", "fixed"}	Specifica la politica di allocazione della cella secondaria
payload_size	`int`	$[0, \infty)$	Dimensione del pacchetto di dato
ie_payload_size	`int`	$[0, \infty)$	Dimensione dell'array scambiato

update_period	`int`	$[0, \infty)$	Periodicità dell'aggiornamento
alpha	`float`	$[0, 1]$	

Tabella 4.3: Elenco dei parametri supportati dal simulatore

Capitolo 5

Risultati

5.1 Setup e metriche di interesse

Per valutare l'efficacia delle soluzioni e dei protocolli proposti in questo lavoro di tesi è stata svolta una campagna di simulazioni orientata alla raccolta di informazioni rilevanti riguardanti il consumo energetico dei nodi, le latenze delle trasmissioni ed altre metriche di interesse, nel seguito descritte.

La topologia della rete simulata consiste semplicemente in una coppia di nodi, N_{TX} e N_{RX}, collegati direttamente tra loro per mezzo di una singola cella TSCH principale, a cui si somma una cella di backup da utilizzare nelle transazioni CONSIP. Tale topologia è raffigurata in Figura 5.1 insieme con la rappresentazione dei due flussi di informazione che transitano da N_{TX} a N_{RX}: il flusso applicativo, rappresentato in blu e caratterizzato da un periodo `T_app` pari a 60 s comune a tutte le simulazioni e da una dimensione pari a `packet_size`, e il flusso associato alla trasmissione della configurazione di rete aggiornata, rappresentato in arancione e caratterizzato da un periodo `T_update` e da una dimensione dell'aggiornamento ν `update_size`, entrambi variabili.

Lo scopo di ciascuna simulazione è quello di valutare le metriche di interesse al variare di un determinato numero di parametri di configurazione. Nelle simulazioni relative all'analisi del protocollo CONSIP (Sezione 5.2) i due parametri variati sono la periodicità e la dimensione dell'aggiornamento ν, mentre nelle simulazioni relative all'algoritmo di whitelisting probabilistico (Sezione 5.3) i parametri variabili sono nuovamente la periodicità dell'aggiornamento e il parametro α utilizzato in Equazione (3.4). Le metriche di interesse, invece, sono relative alle seguenti caratteristiche della rete: consumo energetico, latenza, quantità di informazione aggiuntiva scambiata e numero di ritrasmissioni. Il simulatore `TSCHpredictor` fornisce, per ogni simulazione, il resoconto energetico dei nodi attraverso le seguenti metriche, tutte espresse in µW:

- $\boldsymbol{P_{tx/tot}^{N_{TX}}}$: quantità totale di energia consumata dal nodo trasmettitore e coincidente con l'energia utilizzata per le sole operazioni di trasmissione dei frame

- $\boldsymbol{P_{rx}^{N_{RX}}}$: quantità di energia consumata dal nodo ricevitore per le operazioni di ricezione dei frame

- $\boldsymbol{P_{idle}^{N_{RX}}}$: quantità di energia consumata durante la fase di idle listening. Incide pesantemente sul computo totale di energia consumata dal nodo ed è, in quantità variabile, incrementata dall'introduzione del protocollo CONSIP

- $\boldsymbol{P_{tot}^{N_{RX}}}$: quantità di energia totale consumata dal nodo trasmettitore. Somma delle due quantità precedenti

- $\boldsymbol{P_{tot}}$: somma dell'energia consumata globalmente dalla coppia di nodi N_{TX} e N_{RX}. Viene riportata nelle tabelle successive anche sotto forma di incremento percentuale rispetto all'energia totale misurata nella simulazione corrispondente alla disattivazione rispettivamente del protocollo CONSIP e del protocollo di whitelisting probabilistico

Al termine della simulazione il software `TSCHpredictor`, oltre a stampare su schermo le precedenti informazioni, produce un file di log nel quale sono riportati tutti i tentativi di trasmissione dei frame relativi ai rispettivi flussi. Per ogni pacchetto viene riportato il numero di tentativi di trasmissione e la latenza complessiva dello scambio, espressa in secondi. Attraverso l'analisi di tale database è possibile definire le metriche di interesse relative alla latenza complessiva della comunicazione: media aritmetica ($\boldsymbol{\mu_d}$), deviazione standard ($\boldsymbol{\sigma_d}$), 99 percentile ($\boldsymbol{d_{p99}}$) e 99.9 percentile ($\boldsymbol{d_{p99.9}}$)[1].

Come visto in Sezione 2.2, l'informazione aggiuntiva ν viene inserita nell'header dei frame di dato che transitano dal nodo trasmettitore al nodo ricevitore. Tale configurazione, dunque, non viene esplicitamente trasmessa nell'istante di generazione ma viene inserita all'interno di appositi campi nel primo messaggio utile presente nella coda di trasmissione. È possibile distinguere due tipologie di messaggi: $\mathcal{F}$ e $\mathcal{F}_\nu$. Il primo rappresenta un semplice frame di dato, generato con una determinata periodicità `T_app`, mentre il secondo rappresenta un frame di dato a cui viene concatenata la configurazione aggiornata. La periodicità `T_update` dell'aggiornamento è oggetto di analisi in Sezione 5.2. Come anticipato in precedenza, si è interessati a valutare la percentuale di quantità di informazione aggiuntiva scambiata e, a tale scopo, viene definita e analizzata la quantità $\mathbf{n_{update}} = \dfrac{\mathcal{F}_\nu}{\mathcal{F}_\nu + \mathcal{F}}$.

[1] È stata utilizzata la lettera d in quanto iniziale del termine *delay*

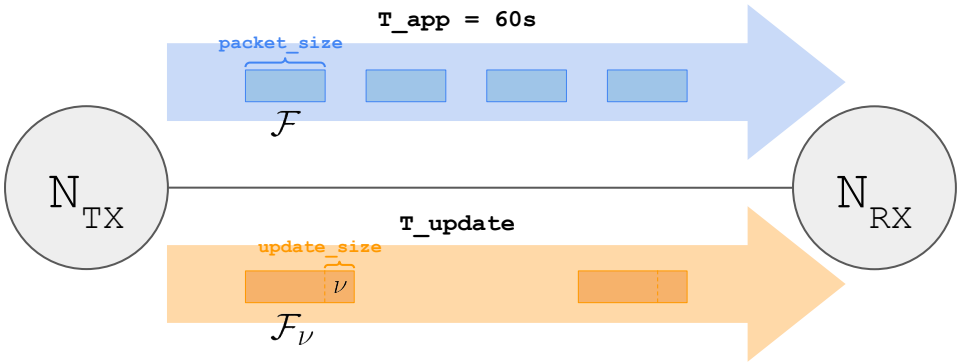


Figura 5.1: Topologia utilizzata nelle simulazioni

5.2 Simulazioni protocollo CONSIP

Il primo gruppo di simulazioni ha lo scopo di valutare l'incidenza sul piano energetico del protocollo CONSIP. Ricordando che lo scopo del protocollo è quello di garantire la coerenza dell'informazione ai due estremi della comunicazione rimediando a possibili errori di trasmissione sul link, è possibile fare delle ipotesi sull'*overhead* energetico introdotto. Il nodo trasmettitore, in ogni istante, mantiene attiva una sola delle due celle disponibili (C_{main}) e dunque, nel caso in cui la comunicazione sia unidirezionale come avviene nelle simulazioni proposte, non è prevista alcuna fase di double listening o di idle listening (Figura 2.5). Ci si può aspettare un consumo energetico in linea con quello misurato in assenza di ν-exchange. Il nodo ricevitore, invece, è di per sé soggetto al fenomeno di idle listening, in quanto non è possibile stabilire in anticipo se il nodo trasmettitore abbia intenzione di trasmettere informazione o meno. Questo comporta un notevole incremento nel consumo di energia poiché il nodo è costretto a porsi in fase di ascolto in corrispondenza di ogni cella attiva schedulata. A tal proposito, la regolazione della lunghezza del timeslot consente di monitorare il consumo energetico del nodo ricevitore: una maggiore durata dello stesso provoca una maggiore latenza tra due fasi di ascolto successive a vantaggio di una maggiore autonomia del dispositivo ma a discapito della reattività della rete e della latenza minima garantita per lo scambio di frame. In aggiunta a questo, CONSIP prevede una fase di double listening che, seppur limitata nel tempo, causa un consumo di energia all'incirca doppio durante tali intervalli.

Le simulazioni presentate in questa sezione sono ottenute variando i valori di `T_update` e `dim` del vettore ν. La prima esecuzione del simulatore, inoltre, è eseguita con l'algoritmo CONSIP disabilitato in modo da disporre di un metro di paragone con le altre simulazioni. In Tabella 5.1 è stato definito un modello di canale a probabilità costanti. I valori utilizzati sono rispettivamente FDP = 0.874 e ACKDP = 0.92 e sono stati ricavati dal monitoraggio di uno scenario WSN reale

[38]. Gli altri parametri, costanti e utilizzati anche per le simulazioni successive, laddove non espressamente indicato un diverso valore, sono:

- dimensione del frame di dato (senza aggiornamento) pari a 61 B
- periodicità del flusso pari a 60 s
- slotframe di 101 slot e 16 canali a disposizione
- numero massimo di trasmissioni pari a 16
- politica di collocamento della cella secondaria equispaziata. Nella pratica si utilizzano le celle allo slot offset 1 e 51

La scelta dei valori di periodicità dell'aggiornamento deriva direttamente dal numero minimo di campioni a disposizione da parte dell'algoritmo di monitoraggio dello stato del canale per effettuare un calcolo opportuno del vettore PDR (Equazione (3.3)). Posti T_app = 1 min e $N_{ch} \equiv |\mathcal{C}| = 16$, la periodicità minima dell'aggiornamento ν affinché venga preso in considerazione, in media, un campione per ogni canale è pari a $N_{ch} \cdot$ T_app = 16 min. In tabella, per utilizzare valori multipli e sottomultipli dell'ora, si è deciso di porre tale limite inferiore al valore di 15 min. La dimensione dell'aggiornamento, invece, varia da un massimo di 16 B, nel quale ogni campione utilizza 8 bit, ad un minimo di 8 B, risultato, ad esempio, di eventuali ottimizzazioni nella codifica dell'informazione. Tali valori sono stati selezionati comunque in maniera arbitraria e sono possibili dimensioni maggiori o minori rispetto a quelle riportate.

Analizzando i risultati in Tabella 5.1 è subito possibile validare l'assunzione fatta in precedenza relativa al minor consumo energetico totale P_{tot} della soluzione con CONSIP disabilitato rispetto alle altre simulazioni. I valori relativi al consumo totale di N_{TX}, nella colonna $P_{tx/tot}^{N_{TX}}$, sono leggermente superiori nelle simulazioni con ridotta periodicità dell'aggiornamento. Tale risultato è in linea con le aspettative, poiché la frequenza della trasmissione di un frame di dimensioni maggiori – secondo quanto espresso dal modello energetico contenuto in [43] e visto in Sezione 4.3.2 – è sempre minore all'aumentare di T_{update} e il peso relativo dell'informazione aggiuntiva tende ad essere trascurabile. Nel caso di $T_{update} = 15$ min, inoltre, è possibile notare una leggera differenza nella quantità di energia necessaria a trasmettere informazione aggiuntiva di dimensione differente. La costanza dei valori nei casi successivi suggerisce di evitare ottimizzazioni eccessive nella codifica del vettore di configurazione. Le differenze più evidenti si possono notare sul nodo N_{RX} ed in particolare in corrispondenza del valore $P_{idle}^{N_{RX}}$. La frequenza dei periodi di double listening, che, come detto, aumentano i consumi energetici, e l'incidenza dal punto di vista energetico degli stessi sono inversamente proporzionali al periodo di generazione e trasmissione dell'aggiornamento della configurazione. Nel caso minimale, con $T_{update} = 15$ min, si verifica un incremento percentuale totale di potenza pari

		Consumo energetico						Latenza				
T_{update}	dim	$P_{\text{tx/tot}}^{N_{TX}}$	$P_{\text{rx}}^{N_{RX}}$	$P_{\text{idle}}^{N_{RX}}$	$P_{\text{tot}}^{N_{RX}}$	P_{tot}		μ_d	σ_d	d_{p99}	$d_{p99.9}$	n_{update}
[min]	[B]	[µW]	[µW]			[µW]	[%]	[s]				[%]
disabilitato		4.31	4.91	65.46	70.37	74.68	-	1.31	1.01	4.92	7.24	-
15	8	4.33	4.93	70.01	74.94	79.27	+6.15%	1.31	1.01	4.88	7.22	+7.18%
	10	4.34	4.93	70.01	74.94	79.28	+6.16%	1.31	1.01	4.88	7.22	+7.18%
	12	4.35	4.93	70.01	74.94	79.29	+6.17%	1.31	1.01	4.88	7.22	+7.18%
	16	4.36	4.94	70.01	74.95	79.31	+6.20%	1.31	1.01	4.88	7.22	+7.18%
30	8	4.32	4.92	67.73	72.65	76.97	+3.07%	1.31	1.01	4.90	7.20	+3.47%
	10	4.33	4.92	67.73	72.65	76.98	+3.08%	1.31	1.01	4.90	7.20	+3.47%
	12	4.33	4.92	67.73	72.65	76.98	+3.09%	1.31	1.01	4.90	7.20	+3.47%
	16	4.33	4.93	67.73	72.66	76.99	+3.10%	1.31	1.01	4.90	7.20	+3.47%
60	8	4.32	4.91	66.59	71.51	75.83	+1.54%	1.31	1.01	4.90	7.24	+1.72%
	10	4.32	4.92	66.59	71.51	75.83	+1.54%	1.31	1.01	4.90	7.24	+1.72%
	12	4.32	4.92	66.59	71.51	75.83	+1.54%	1.31	1.01	4.90	7.24	+1.72%
	16	4.32	4.92	66.59	71.51	75.84	+1.55%	1.31	1.01	4.90	7.24	+1.72%
120	8	4.31	4.91	66.02	70.94	75.25	+0.77%	1.31	1.01	4.90	7.24	+0.85%
	10	4.32	4.91	66.02	70.94	75.25	+0.77%	1.31	1.01	4.90	7.24	+0.85%
	12	4.32	4.91	66.02	70.94	75.26	+0.77%	1.31	1.01	4.90	7.24	+0.85%
	16	4.32	4.91	66.02	70.94	75.26	+0.77%	1.31	1.01	4.90	7.24	+0.85%
240	8	4.31	4.91	65.74	70.65	74.97	+0.38%	1.31	1.01	4.90	7.24	+0.42%
	10	4.31	4.91	65.74	70.65	74.97	+0.38%	1.31	1.01	4.90	7.24	+0.42%
	12	4.31	4.91	65.74	70.65	74.97	+0.39%	1.31	1.01	4.90	7.24	+0.42%
	16	4.31	4.91	65.74	70.65	74.97	+0.39%	1.31	1.01	4.90	7.24	+0.42%
480	8	4.31	4.91	65.60	70.51	74.82	+0.19%	1.31	1.01	4.90	7.20	+0.21%
	10	4.31	4.91	65.60	70.51	74.82	+0.19%	1.31	1.01	4.90	7.20	+0.21%
	12	4.31	4.91	65.60	70.51	74.82	+0.19%	1.31	1.01	4.90	7.20	+0.21%
	16	4.31	4.91	65.60	70.51	74.82	+0.19%	1.31	1.01	4.90	7.20	+0.21%

Tabella 5.1: Simulazione CONSIP con probabilità costanti

al 6% circa, mentre nel caso immediatamente successivo tale valore decresce al 3% circa. In entrambi i casi è stato possibile garantire un ν-exchange coerente con un incremento di energia tutto sommato accettabile in numerosi contesti applicativi. L'ulteriore riduzione del periodo consente di raggiungere valori trascurabili. Ciò dimostra che lo svantaggio principale del protocollo CONSIP non è l'incremento dei consumi ma la necessità di raddoppiare il numero di celle e, di conseguenza, dimezzare la capacità dello slotframe. Future ottimizzazioni del protocollo potrebbero prevedere una riduzione del numero di celle di backup.

Le colonne relative alla latenza rispettano le previsioni, dato che il protocollo di per sé non introduce una funzione di whitelisting e, dunque, non applica una reale modifica alla configurazione dei canali. Altre metriche interessanti sono quelle relative alle latenze del protocollo CONSIP, calcolate a partire dai quattro istanti temporali che caratterizzano il ν-exchange:

- t_{UR} (*update request*) istante in cui l'algoritmo di aggiornamento della configurazione genera il nuovo valore di ν. Tale istante è indipendente dalla periodicità delle celle schedulate
- t_{SW} (*switch*) istante in cui il nodo trasmettitore, ricevuto il frame di ACK relativo a $\mathcal{F}_\nu$, effettua lo scambio dello stato delle celle di trasmissione e configura la nuova cella C_{main} per trasmettere utilizzando esclusivamente la nuova configurazione
- t_{DL} (*double listening*) istante in cui il nodo ricevitore riceve correttamente $\mathcal{F}_\nu$. Da questo momento in poi entrambe le celle C_{main} e C_{backup} vengono poste in modalità double listening
- t_E (*end*) istante in cui N_{RX} riceve correttamente un frame di dato nella cella C_{backup}, ossia utilizzando la nuova configurazione. In questo momento la fase di double listening viene interrotta e il ν-exchange può ritenersi concluso

In Figura 5.2 è possibile visualizzare questi istanti temporali in un contesto realistico di ν-exchange. Per la raccolta di tali campioni temporali il simulatore `TSCHpredictor` è stato leggermente modificato e il file di output corrispondente alle latenze dell'algoritmo è stato analizzato e i risultati sono mostrati in Tabella 5.2.

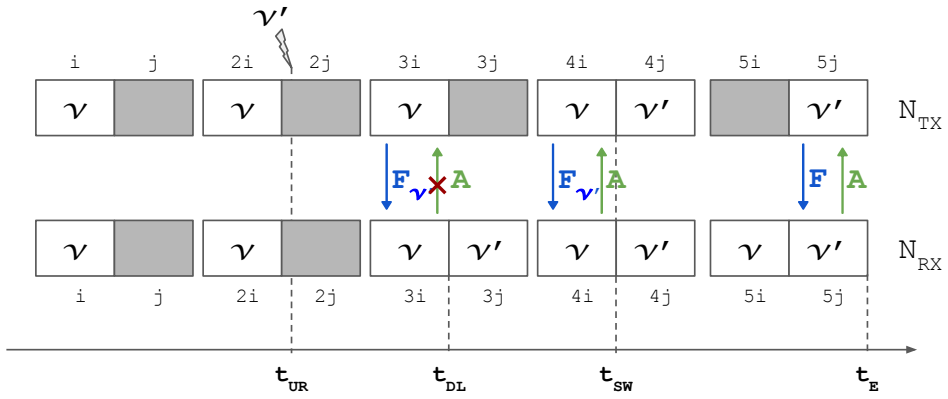


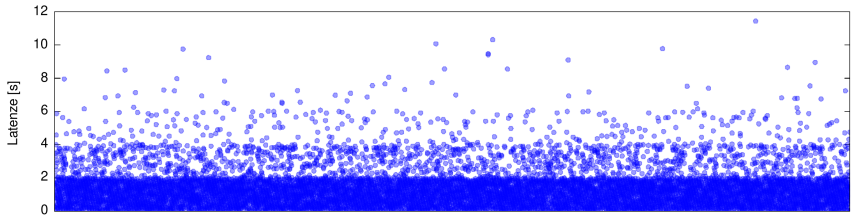
Figura 5.2: Istanti temporali in CONSIP

Le tre latenze analizzate sono $d_{SW} = t_{SW} - t_{UR}$, $d_{DL} = t_E - t_{DL}$ e $d_{tot} = t_E - t_{UR}$. La quantità d_{SW} indica la latenza tra l'istante di generazione di una nuova configurazione e l'istante di effettivo utilizzo della stessa da parte del nodo trasmettitore. Ricordando che la periodicità di una singola cella è pari a 2 s, la media aritmetica e la deviazione standard misurate mostrano come tale latenza sia confrontabile con l'intervallo temporale che intercorre tra due celle consecutive dello stesso tipo. Poiché l'evento di aggiornamento (t_{UR}) è indipendente dalla posizione delle celle è

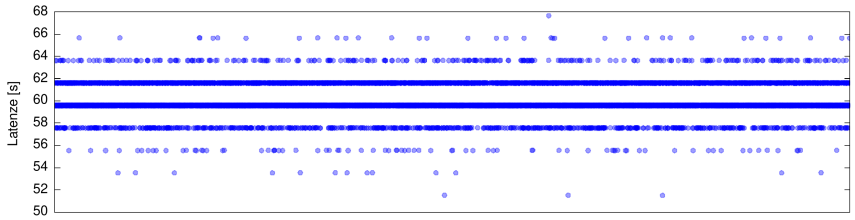
Latenze algoritmo	μ_d	σ_d	d_{min}	d_{max}
	[s]			
$d_{SW} = t_{SW} - t_{UR}$	1.47	1.24	0.00	11.44
$d_{DL} = t_E - t_{DL}$	60.01	1.55	51.50	67.66
$d_{tot} = t_E - t_{UR}$	61.30	1.20	60.00	68.32

Tabella 5.2: Latenze algoritmo

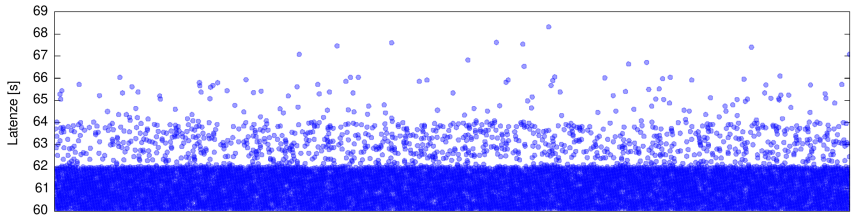
logico aspettarsi, in media, un valore di latenza d_{SW} compreso tra 0 e 2 s. Il valore d_{DL}, invece, è relativo alla durata della fase di double listening per ogni transazione CONSIP. Tale fase si conclude con la ricezione del frame successivo a quello che ne ha segnato l'inizio e, di conseguenza, il valore di media aritmetica espresso in tabella è in linea con le aspettative. Con tempi di simulazione sufficientemente lunghi, infatti, la durata media del double listening tende al valore di periodicità del flusso di dati. La latenza totale d_{tot}, invece, è semplicemente data dalla somma delle due



(a) d_{SW}



(b) d_{DL}



(c) d_{tot}

Figura 5.3: Grafico a dispersione latenze CONSIP. I risultati sono ottenuti in seguito ad una simulazione `TSCHpredictor` di durata pari ad 1 anno

precedenti e rappresenta l'intera latenza di un singolo ν-exchange. In Figura 5.3 vengono rappresentati i tre grafici a dispersione relativi alle latenze appena descritte. È interessante, infine, sottolineare la proporzionalità tra l'aumento dei consumi energetici e la percentuale di pacchetti trasmessi contenenti il valore aggiornato ν. Nel grafico in Figura 5.4 è possibile visualizzare tale correlazione.

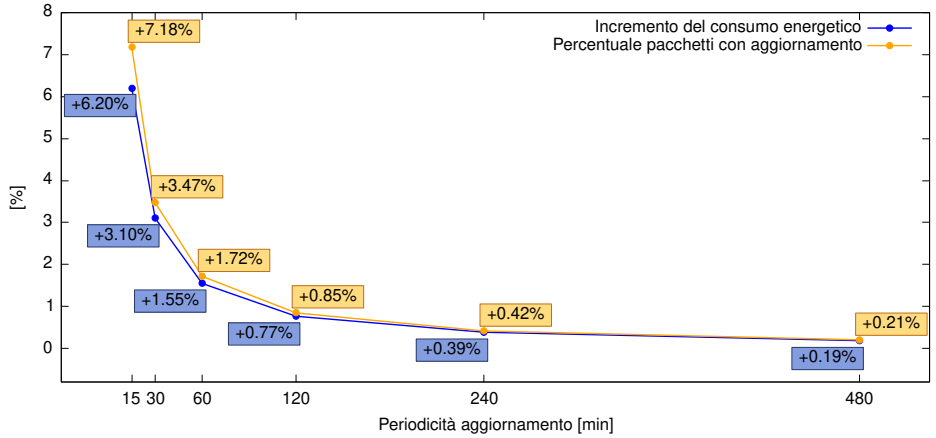


Figura 5.4: Correlazione tra consumo energetico e percentuale di pacchetti contenenti l'aggiornamento

La seconda simulazione è basata, invece, su un modello di canale acquisito tramite il monitoraggio di uno scenario di laboratorio reale per mezzo degli strumenti descritti in Sezione 4.2. Il database che ne deriva è relativo ad un intervallo tem-

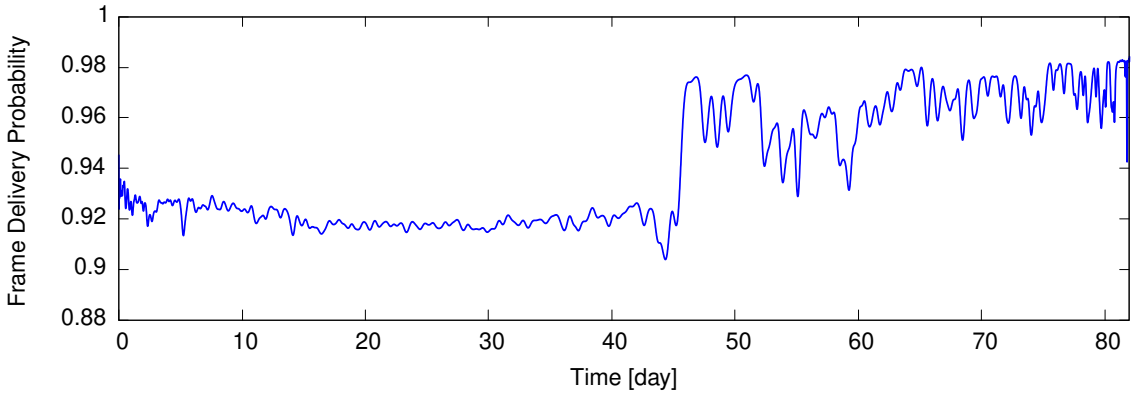


Figura 5.5: Modello di canale reale utilizzato per le simulazioni

porale di 82 giorni consecutivi, durante i quali il livello misurato di qualità della comunicazione è relativamente alto. I valori medi ottenuti a fronte di un numero di frame trasmessi superiore a 100 milioni sono $\text{FDP}_{\text{avg}} = 0.94$ e $\text{ACKDP}_{\text{avg}} = 0.96$.

Questi valori sono giustificati principalmente dai seguenti due fattori: la vicinanza relativa tra i due nodi e l'assenza prolungata di importanti interferenze esterne dovuta a periodi di chiusura, anche per diversi giorni consecutivi, del laboratorio. In Figura 5.5 viene tracciata l'evoluzione temporale del solo valore di FDP. La finestra mobile utilizzata è caratterizzata da un numero di campioni pari a 5000 unità e da uno scostamento relativo tra misure successive pari a 500 unità. In tale grafico è possibile notare un incremento medio del tasso di successo di trasmissione dei frame in corrispondenza del 45esimo giorno circa, seppur limitato a pochi punti percentuali. Il file contenente i campioni acquisiti dal monitoraggio è stato utilizzato per

		Consumo energetico						Latenza				
T_{update}	dim	$P_{\text{tx/tot}}^{N_{TX}}$	$P_{\text{rx}}^{N_{RX}}$	$P_{\text{idle}}^{N_{RX}}$	$P_{\text{tot}}^{N_{RX}}$	P_{tot}		μ_d	σ_d	d_{p99}	$d_{p99.9}$	n_{update}
[min]	[B]	[µW]		[µW]		[µW]	[%]			[s]		[%]
disabilitato		3.90	4.55	65.73	70.28	74.18	-	1.16	0.88	3.90	6.02	-
15	8	3.91	4.56	70.29	74.85	78.76	+6.18%	1.16	0.88	3.90	6.06	+7.08%
	10	3.92	4.56	70.29	74.86	78.77	+6.19%	1.16	0.88	3.90	6.06	+7.08%
	12	3.92	4.56	70.29	74.86	78.78	+6.21%	1.16	0.88	3.90	6.06	+7.08%
	16	3.93	4.57	70.29	74.87	78.80	+6.23%	1.16	0.88	3.90	6.06	+7.08%
30	8	3.91	4.56	68.00	72.57	76.48	+3.11%	1.16	0.88	3.90	6.54	+3.45%
	10	3.92	4.56	68.00	72.57	76.49	+3.11%	1.16	0.88	3.90	6.54	+3.45%
	12	3.92	4.57	68.00	72.57	76.49	+3.12%	1.16	0.88	3.90	6.54	+3.45%
	16	3.92	4.57	68.00	72.57	76.50	+3.13%	1.16	0.88	3.90	6.54	+3.45%
60	8	3.90	4.56	66.87	71.42	75.33	+1.56%	1.16	0.89	3.92	6.76	+1.68%
	10	3.91	4.56	66.87	71.43	75.33	+1.56%	1.16	0.89	3.92	6.76	+1.68%
	12	3.91	4.56	66.87	71.43	75.33	+1.56%	1.16	0.89	3.92	6.76	+1.68%
	16	3.91	4.56	66.87	71.43	75.34	+1.57%	1.16	0.89	3.92	6.76	+1.68%
120	8	3.90	4.56	66.30	70.85	74.76	+0.79%	1.16	0.89	3.92	6.06	+0.83%
	10	3.90	4.56	66.30	70.85	74.76	+0.79%	1.16	0.89	3.92	6.06	+0.83%
	12	3.91	4.56	66.30	70.86	74.76	+0.79%	1.16	0.89	3.92	6.06	+0.83%
	16	3.91	4.56	66.30	70.86	74.76	+0.79%	1.16	0.89	3.92	6.06	+0.83%
240	8	3.90	4.56	66.01	70.57	74.47	+0.40%	1.16	0.89	3.90	6.18	+0.42%
	10	3.90	4.56	66.01	70.57	74.47	+0.40%	1.16	0.89	3.90	6.18	+0.42%
	12	3.90	4.56	66.01	70.57	74.47	+0.40%	1.16	0.89	3.90	6.18	+0.42%
	16	3.90	4.56	66.01	70.57	74.47	+0.40%	1.16	0.89	3.90	6.18	+0.42%
480	8	3.90	4.55	65.87	70.43	74.32	+0.20%	1.16	0.90	3.90	6.44	+0.22%
	10	3.90	4.55	65.87	70.43	74.32	+0.20%	1.16	0.90	3.90	6.44	+0.22%
	12	3.90	4.55	65.87	70.43	74.32	+0.20%	1.16	0.90	3.90	6.44	+0.22%
	16	3.90	4.55	65.87	70.43	74.33	+0.20%	1.16	0.90	3.90	6.44	+0.22%

Tabella 5.3: Simulazione CONSIP con probabilità acquisite con OpenWSN

pilotare il secondo gruppo di simulazioni, il cui risultato è mostrato in Tabella 5.3. In breve, le conclusioni tratte nell'analisi del gruppo di simulazioni precedente si applicano anche a questa tabella. Gli incrementi percentuali di energia totale e la percentuale di pacchetti contenenti informazione aggiuntiva restano pressoché

invariati, con l'unica differenza che, a causa della maggior qualità dei link, i valori assoluti di energia consumata sono leggermente inferiori.

5.3 Simulazioni protocollo di whitelisting probabilistico

Dopo aver analizzato il profilo energetico di una rete coinvolta in un semplice scambio CONSIP periodico è stato eseguito un secondo gruppo di simulazioni orientato a valutare le caratteristiche del protocollo di whitelisting probabilistico. Secondo quanto visto nella sezione precedente la variazione della dimensione dell'aggiornamento ν non introduce differenze sostanziali nell'incremento complessivo di potenza consumata. Per questo motivo si è scelto di fissare la dimensione dell'aggiornamento al valore dim $= 16\,\mathrm{B}$. I parametri variabili, dunque, sono la periodicità dell'aggiornamento stesso T_{update} e il valore assunto dal parametro α. Ricordando che quest'ultimo definisce il peso relativo dell'ultima finestra temporale nel calcolo del vettore PDR aggiornato (Equazione (3.4)), i valori selezionati variano dal minimo $\alpha = 0.2$ al massimo $\alpha = 1$, nel quale l'aggiornamento ν rappresenta esclusivamente lo stato del canale nell'ultimo Δt di riferimento.

Rispetto alla sezione precedente non è presente il gruppo di simulazioni relativo a probabilità di successo costanti in quanto, in tal caso, l'algoritmo di whitelisting risulterebbe inutile poiché il vettore ν sarebbe costante nel tempo e non applicherebbe modifiche alla configurazione di rete dei due nodi. È necessario valutare l'algoritmo sfruttando dei modelli di canale variabili nel tempo. In Tabella 5.4 sono riportati i risultati delle simulazioni caratterizzate da probabilità di successo reali e, come visto in precedenza, pari a $\mathrm{FDP}_{\mathrm{avg}} = 0.94$ e $\mathrm{ACKDP}_{\mathrm{avg}} = 0.96$. È possibile ritrovare valori di incremento percentuale della potenza e di pacchetti $\mathcal{F}_\nu$ rispetto al totale comparabili con quelli delle simulazioni precedenti. Anche in questo caso le latenze non sono peggiorate, a suggerimento del fatto che nel peggiore degli scenari tale protocollo non introduce peggioramenti ma, al contrario, ci si possono aspettare metriche relative alla latenza più ridotte o, al più, simili rispetto alla simulazione con l'algoritmo disabilitato. Nel caso specifico non sono visibili migliorie evidenti in quanto il modello del canale utilizzato ha un livello di qualità ottimo. Questo ci consente di stabilire che l'applicazione dell'algoritmo di whitelisting dev'essere attentamente valutata ed introdotta solamente in uno scenario di rete soggetto ad un elevato tasso di errore di trasmissione per un sottoinsieme di canali utilizzabili. Solo in quel caso l'incremento del consumo energetico sarebbe giustificato da un miglioramento in termini di affidabilità e robustezza della rete. Nonostante ciò, è possibile iniziare ad intravedere delle leggere variabilità nei parametri di latenza misurati tra i vari scenari. Queste differenze sono maggiormente visibili nel secondo gruppo di simulazioni, nel quale si è utilizzato un modello di canale artificiale e caratterizzato dalla presenza di interferenze prolungate e distruttive che provocano un

T_{update} [min]	α	Consumo energetico $P^{N_{TX}}_{\text{tx/tot}}$ [µW]	$P^{N_{RX}}_{\text{rx}}$ [µW]	$P^{N_{RX}}_{\text{idle}}$ [µW]	$P^{N_{RX}}_{\text{tot}}$ [µW]	P_{tot} [µW]	[%]	Latenza μ_d [s]	σ_d [s]	d_{p99} [s]	$d_{p99.9}$ [s]	n_{update} [%]
disabilitato		3.89	70.28	65.73	4.55	74.18	-	1.17	0.92	3.90	6.64	-
15	0.2	3.91	74.86	70.30	4.56	78.77	+6.20%	1.16	0.91	3.90	6.60	+7.16%
	0.4	3.91	74.86	70.31	4.56	78.77	+6.19%	1.16	0.91	3.86	6.52	+7.06%
	0.5	3.91	74.86	70.30	4.55	78.77	+6.19%	1.16	0.93	3.90	6.78	+7.10%
	0.6	3.91	74.86	70.31	4.55	78.77	+6.19%	1.16	0.92	3.88	6.86	+7.13%
	0.8	3.91	74.86	70.30	4.55	78.77	+6.19%	1.16	0.91	3.88	6.78	+7.16%
	1.0	3.90	74.86	70.31	4.55	78.76	+6.18%	1.16	0.91	3.88	6.66	+7.13%
30	0.2	3.90	72.57	68.02	4.55	76.47	+3.10%	1.17	0.94	3.92	7.04	+3.44%
	0.4	3.89	72.57	68.02	4.54	76.46	+3.08%	1.16	0.90	3.88	6.20	+3.44%
	0.5	3.90	72.57	68.02	4.54	76.46	+3.08%	1.16	0.92	3.88	6.40	+3.45%
	0.6	3.90	72.57	68.02	4.55	76.47	+3.09%	1.16	0.92	3.88	7.18	+3.46%
	0.8	3.89	72.57	68.02	4.54	76.46	+3.08%	1.16	0.93	3.88	6.68	+3.46%
	1.0	3.90	72.57	68.02	4.55	76.47	+3.08%	1.16	0.92	3.88	6.38	+3.48%
60	0.2	3.88	71.42	66.89	4.54	75.31	+1.52%	1.16	0.92	3.90	6.72	+1.69%
	0.4	3.88	71.42	66.89	4.54	75.31	+1.53%	1.16	0.91	3.88	6.68	+1.67%
	0.5	3.89	71.42	66.88	4.54	75.31	+1.53%	1.16	0.91	3.90	6.50	+1.71%
	0.6	3.89	71.42	66.88	4.54	75.31	+1.53%	1.16	0.92	3.90	6.54	+1.69%
	0.8	3.88	71.42	66.89	4.54	75.31	+1.52%	1.16	0.91	3.86	6.60	+1.69%
	1.0	3.88	71.42	66.89	4.54	75.31	+1.52%	1.16	0.92	3.90	6.42	+1.70%
120	0.2	3.89	70.85	66.31	4.54	74.74	+0.76%	1.16	0.92	3.88	6.78	+0.84%
	0.4	3.88	70.85	66.31	4.54	74.73	+0.75%	1.16	0.92	3.90	6.42	+0.84%
	0.5	3.89	70.85	66.31	4.54	74.74	+0.76%	1.16	0.93	3.90	6.62	+0.84%
	0.6	3.88	70.85	66.32	4.53	74.73	+0.74%	1.16	0.92	3.90	6.82	+0.84%
	0.8	3.88	70.85	66.32	4.53	74.73	+0.74%	1.16	0.90	3.88	6.20	+0.83%
	1.0	3.88	70.85	66.31	4.54	74.73	+0.75%	1.16	0.92	3.90	6.52	+0.86%
240	0.2	3.88	70.57	66.03	4.54	74.45	+0.37%	1.16	0.92	3.90	6.74	+0.42%
	0.4	3.89	70.57	66.02	4.54	74.45	+0.37%	1.16	0.93	3.90	7.04	+0.42%
	0.5	3.88	70.56	66.03	4.53	74.44	+0.36%	1.16	0.90	3.90	6.86	+0.42%
	0.6	3.88	70.57	66.03	4.54	74.45	+0.37%	1.16	0.92	3.90	6.70	+0.42%
	0.8	3.88	70.56	66.03	4.54	74.44	+0.36%	1.16	0.91	3.88	6.52	+0.41%
	1.0	3.88	70.56	66.03	4.54	74.44	+0.36%	1.16	0.91	3.90	6.32	+0.42%
480	0.2	3.89	70.42	65.88	4.54	74.31	+0.18%	1.17	0.92	3.92	7.02	+0.21%
	0.4	3.88	70.42	65.88	4.54	74.31	+0.18%	1.16	0.92	3.90	6.96	+0.21%
	0.5	3.88	70.42	65.89	4.53	74.30	+0.17%	1.16	0.89	3.90	6.86	+0.21%
	0.6	3.88	70.42	65.89	4.54	74.30	+0.17%	1.16	0.91	3.88	6.74	+0.21%
	0.8	3.88	70.42	65.89	4.53	74.30	+0.16%	1.16	0.91	3.90	6.60	+0.21%
	1.0	3.88	70.42	65.88	4.54	74.31	+0.17%	1.16	0.91	3.90	6.64	+0.21%

Tabella 5.4: Simulazione algoritmo di whitelisting probabilistico con probabilità acquisite con OpenWSN

peggioramento drastico di FDP in corrispondenza di specifici gruppi di canali. In Figura 5.6 è possibile visualizzare il modello utilizzato. Il valore di ACKDP, invece, resta costante per tutto il tempo della simulazione e assume il valore, condiviso da

tutti i canali, di 0.92. Rispetto alle precedenti simulazioni, il valore di periodicità

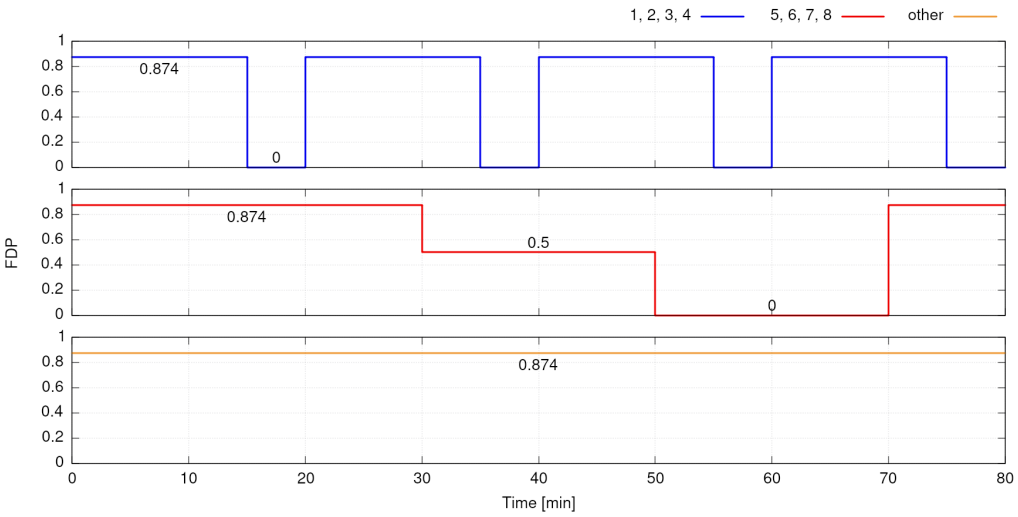


Figura 5.6: Modello di canale utilizzato per la simulazione

del flusso dati T_app è stato ridotto a 5 s e, di conseguenza, la periodicità minima di aggiornamento della configurazione è stata portata a $T_{update,\,min} = 1\,\text{min}$. Con tale valore l'algoritmo di aggiornamento di ν ha a disposizione in media 0.75 campioni per canale per valutare il PDR nella finestra temporale Δt trascorsa. I valori successivi sono stati selezionati secondo un fattore moltiplicativo pari a 2 rispetto alla simulazione precedente. Nonostante il modello utilizzato sia molto semplificato e, allo stesso tempo, soggetto ad interferenze distruttive che rendono impossibile la comunicazione su alcuni canali per determinati intervalli di tempo, è di particolare interesse l'analisi dei risultati mostrati in Tabella 5.5. Innanzitutto, a causa della maggiore frequenza di trasmissione dei frame, i valori assoluti delle diverse componenti di consumo energetico dei nodi sono molto più elevati. È subito possibile notare, però, come l'introduzione dell'algoritmo di whitelisting abbia introdotto dei vantaggi non trascurabili in termini di latenza della comunicazione, raggiungendo l'obiettivo principale di tale protocollo. I valori di media aritmetica raccolti registrano una diminuzione che varia dal 25% al 50% rispetto alla simulazione con l'algoritmo disattivato. In generale, la riduzione di tali metriche conferma come l'algoritmo di whitelisting sia efficace in uno scenario in cui siano presenti interferenze non trascurabili dal punto di vista dell'intensità e dell'estensione temporale. Per ogni gruppo di simulazioni viene evidenziata la configurazione di α più favorevole. In Figura 5.7 sono mostrati gli andamenti di μ_d al variare di α e di T_{update}. Nello scenario proposto, nella maggior parte dei casi, la scelta di un valore α prossimo ad 1 ha fornito i risultati migliori, in quanto la nuova configurazione ν è in grado

T_{update} [min]	α	Consumo energetico $P^{N_{TX}}_{\text{tx/tot}}$ [µW]	$P^{N_{RX}}_{\text{rx}}$ [µW]	$P^{N_{RX}}_{\text{idle}}$ [µW]	$P^{N_{RX}}_{\text{tot}}$ [µW]	P_{tot} [µW]	[%]	Latenza μ_d [s]	σ_d [s]	d_{p99} [s]	$d_{p99.9}$ [s]	n_{update} [%]
disabilitato		64.13	67.54	25.77	93.30	157.44	-	3.45	6.17	33.82	61.96	-
1	0.2	59.34	64.18	35.52	99.70	159.04	+1.02%	1.94	2.03	10.12	17.70	+10.56%
	0.4	57.65	63.00	36.52	99.52	157.17	-0.17%	1.78	1.77	8.72	15.48	+10.19%
	0.5	57.07	62.61	36.89	99.50	156.57	-0.55%	1.73	1.68	8.18	14.98	+10.17%
	0.6	56.66	62.32	37.13	99.46	156.11	-0.84%	1.69	1.62	7.88	14.66	+10.18%
	0.8	55.96	61.84	37.56	99.40	155.36	-1.32%	1.65	1.60	7.72	14.96	+10.04%
	1.0	55.71	61.67	37.73	99.40	155.11	-1.48%	1.65	1.83	7.82	17.92	+10.08%
2	0.2	59.62	64.40	32.04	96.44	156.05	-0.88%	2.04	2.22	11.08	20.06	+4.99%
	0.4	58.32	63.49	32.86	96.35	154.66	-1.76%	1.90	1.98	9.86	17.54	+4.89%
	0.5	57.79	63.13	33.21	96.34	154.13	-2.10%	1.84	1.88	9.36	16.60	+4.93%
	0.6	57.28	62.77	33.52	96.28	153.56	-2.46%	1.80	1.81	8.92	16.66	+4.86%
	0.8	56.51	62.24	34.01	96.25	152.76	-2.97%	1.73	1.73	8.42	15.94	+4.85%
	1.0	56.15	61.99	34.24	96.23	152.38	-3.21%	1.70	1.70	8.18	15.66	+4.83%
4	0.2	60.31	64.87	29.96	94.83	155.14	-1.46%	2.18	2.53	12.74	23.36	+2.51%
	0.4	59.60	64.39	30.44	94.82	154.42	-1.92%	2.09	2.36	11.74	21.54	+2.49%
	0.5	59.20	64.10	30.68	94.78	153.99	-2.19%	2.04	2.25	11.34	19.94	+2.47%
	0.6	58.95	63.93	30.84	94.77	153.71	-2.36%	2.02	2.22	11.20	19.94	+2.48%
	0.8	58.30	63.49	31.27	94.76	153.06	-2.78%	1.93	2.03	10.12	18.28	+2.48%
	1.0	57.89	63.21	31.52	94.73	152.62	-3.06%	1.89	1.98	9.80	18.26	+2.46%
8	0.2	60.69	65.15	28.88	94.03	154.72	-1.72%	2.27	2.77	13.82	27.30	+1.25%
	0.4	60.38	64.92	29.10	94.02	154.40	-1.93%	2.22	2.64	13.14	24.96	+1.28%
	0.5	60.20	64.81	29.20	94.00	154.20	-2.06%	2.19	2.59	12.90	24.24	+1.24%
	0.6	60.01	64.66	29.32	93.98	153.99	-2.19%	2.18	2.61	13.02	25.62	+1.22%
	0.8	59.81	64.53	29.45	93.99	153.80	-2.31%	2.17	2.60	12.94	24.64	+1.22%
	1.0	59.86	64.56	29.42	93.99	153.85	-2.28%	2.19	2.67	13.48	26.20	+1.25%
15	0.2	60.57	65.07	28.57	93.65	154.22	-2.04%	2.26	2.72	13.78	25.48	+0.65%
	0.4	60.72	65.17	28.47	93.64	154.36	-1.95%	2.29	2.83	14.06	28.70	+0.66%
	0.5	60.72	65.17	28.48	93.64	154.36	-1.95%	2.27	2.75	13.88	26.36	+0.63%
	0.6	60.66	65.12	28.52	93.63	154.29	-2.00%	2.26	2.69	13.64	25.24	+0.64%
	0.8	60.32	64.89	28.75	93.64	153.95	-2.21%	2.21	2.59	12.90	24.60	+0.67%
	1.0	59.98	64.65	28.95	93.61	153.59	-2.45%	2.18	2.60	12.78	25.98	+0.63%
30	0.2	60.30	64.89	28.52	93.41	153.72	-2.36%	2.23	2.71	13.56	26.26	+0.29%
	0.4	60.52	65.02	28.38	93.40	153.91	-2.24%	2.28	2.85	14.20	28.24	+0.28%
	0.5	60.67	65.14	28.28	93.42	154.08	-2.13%	2.30	2.84	14.34	27.52	+0.29%
	0.6	60.87	65.28	28.15	93.43	154.30	-1.99%	2.33	2.95	14.78	28.84	+0.30%
	0.8	61.29	65.58	27.87	93.45	154.74	-1.71%	2.43	3.24	16.02	33.96	+0.30%
	1.0	61.78	65.91	27.54	93.46	155.24	-1.39%	2.58	3.73	18.64	40.20	+0.32%

Tabella 5.5: Simulazione algoritmo di whitelisting probabilistico con modello di canale artificiale

di adattarsi con più dinamicità alle forme d'onda riportate nel grafico precedente. La simulazione con $T_{update} = 30\,\text{min}$, tuttavia, presenta un andamento opposto. La motivazione è legata alle periodicità delle forme d'onda in Figura 5.6: l'algoritmo

di aggiornamento entra in azione durante le fasi finali di un'interferenza e produce, per la successiva finestra temporale Δt, una configurazione ν che esclude i canali coinvolti dalla stessa. Durante tale intervallo, tuttavia, l'effetto dell'interferenza scompare ma i nodi sono ancora impossibilitati a trasmettere sullo specifico gruppo di canali che, nel frattempo, ha riacquisito una buona probabilità di successo della trasmissione.

A conclusione, è bene sottolineare come anche i consumi energetici totali possano trarre vantaggio dall'utilizzo dell'algoritmo di whitelisting. Ad eccezione della simulazione con $T_{update} = 1\,\text{min}$ e $\alpha = 0.2$, infatti, la potenza totale complessiva risulta essere minore rispetto alla simulazione di riferimento.

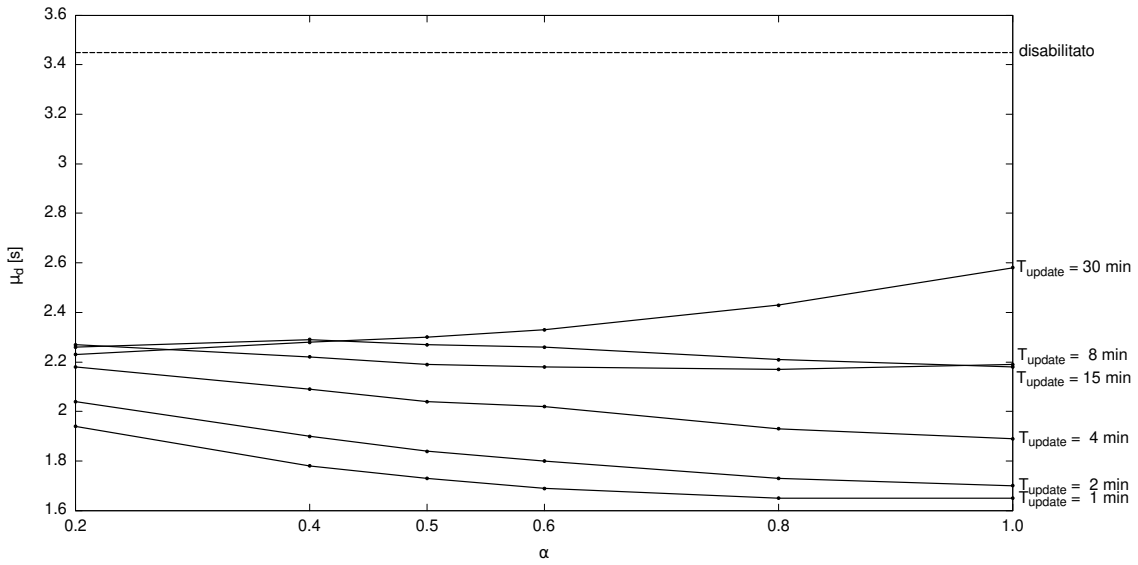


Figura 5.7: Latenza media al variare dei parametri α e T_{update}. Il flusso del traffico dati ha una periodicità T_app = 5 s, mentre il modello di canale utilizzato è quello riportato in Figura 5.6

Conclusioni

In questo lavoro di tesi sono state proposte due tecniche complementari che consentono di limitare la latenza media delle trasmissioni e il numero medio di tentativi di trasmissione dei pacchetti di dato. Per valutarne l'efficacia tali protocolli sono stati implementati su una versione rivista del simulatore ad eventi discreti `TSCHpredictor`. Una fase preliminare di monitoraggio di uno scenario reale di laboratorio e la successiva creazione di un database opportunamente strutturato hanno consentito di estendere la campagna di simulazione oltre l'utilizzo dei soli modelli di canale artificiali. I risultati confermano la robustezza del protocollo CONSIP. A fronte di lievi incrementi nel consumo energetico complessivo, infatti, è stato possibile trasferire gli aggiornamenti della configurazione dei canali ν tra i due nodi senza alcuna perdita di coerenza dell'informazione. L'algoritmo di whitelisting probabilistico, invece, sfruttando il protocollo CONSIP come mezzo per la trasmissione della whitelist, ha introdotto la possibilità di selezionare il canale di trasmissione in maniera differente da quanto descritto nello standard TSCH sfruttando una lista di probabilità dinamiche. In questo caso i risultati dimostrano che tale tecnica non può essere applicata incondizionatamente a tutti gli scenari di rete ma, al contrario, dev'essere introdotta solamente in un contesto di link soggetti a forti interferenze esterne prolungate nel tempo. Futuri lavori potrebbero avere come oggetto la definizione di funzioni di aggiornamento più sofisticate rispetto a quella introdotta in questo lavoro di tesi oppure lo studio di tecniche che consentano di ridurre l'impatto in termini di numero di celle di backup introdotte da CONSIP.

Elenco delle figure

1.1 Trasmissione frame ed eventuale ACK 13
1.2 Esempio di topologia di rete 15
1.3 Esempio di slotframe 16
1.4 Channel Hopping 18
1.5 Bande di frequenza in reti IEEE 802.15.4 e IEEE 802.11 19
1.6 Timer interni al timeslot 21
1.7 Esempio di stack protocollare 23
1.8 Scheduling statico minimale 25
1.9 Scheduling dinamico delle celle 26
1.10 Interazione CoAP client-server con frammentazione 6LoWPAN . . . 30
1.11 Gestione della whitelist in LABeL 33

2.1 Schema di scambio di ν: aggiornamento correttamente effettuato . . 36
2.2 Schema di scambio di ν: perdita non critica di messaggi 37
2.3 Schema di scambio di ν: perdita critica di messaggi 38
2.4 Esempio di slotframe nel protocollo CONSIP 39
2.5 Schema protocollo CONSIP 41
2.6 Macchina a stati protocollo CONSIP 43

3.1 Diagramma a blocchi algoritmo di aggiornamento della configurazione 50

5.1 Topologia utilizzata nelle simulazioni 75
5.2 Istanti temporali in CONSIP 78
5.3 Grafico a dispersione latenze CONSIP 79
5.4 Correlazione tra consumo energetico e percentuale di pacchetti contenenti l'aggiornamento 80
5.5 Modello di canale reale utilizzato per le simulazioni 80
5.6 Modello di canale utilizzato per la simulazione 84
5.7 Latenza media al variare dei parametri α e T_{update} 86

Elenco delle tabelle

4.1 Parametri di configurazione di OpenWSN 58
4.2 Esempio di database generato dal comando `framestats.py store` . 60
4.3 Elenco dei parametri supportati dal simulatore 72

5.1 Simulazione CONSIP con probabilità costanti 77
5.2 Latenze algoritmo 79
5.3 Simulazione CONSIP con probabilità acquisite con OpenWSN . . . 81
5.4 Simulazione algoritmo di whitelisting probabilistico con probabilità acquisite con OpenWSN 83
5.5 Simulazione algoritmo di whitelisting probabilistico con modello di canale artificiale 85

Elenco dei listati

4.1 Caricamento firmware sui dispositivi 52
4.2 Openvisualizer: modulo server 53
4.3 Openvisualizer: modulo client 53
4.4 Campione di pacchetti generati da `packet-gen` 53
4.5 Operazioni di inizializzazione e trasmissione di frame `packet-gen` . . 55
4.6 `framestats.py`: comandi disponibili 59
4.7 Modello di canale variabile 63
4.8 Modello di canale associato a file di campioni 64
4.9 Creazione cella principale e di backup 64
4.10 Implementazione del protocollo CONSIP sul simulatore 66
4.11 Aggiornamento della configurazione `array` 69

Acronimi

6LoWPAN IPv6 over Low-Power Wireless Personal Area Networks.

6TiSCH IPv6 over the TSCH.

ACK ACKnowledge.

ACKDP ACK Delivery Probability.

ASN Absolute Slot Number.

BSN Body Sensor Network.

CCA Clear Channel Assessment.

CoAP Constrained Application Protocol.

CONSIP Consistency Protocol.

CSV Comma Separated Value.

DAO Destination Advertisement Object.

DIO DODAG Information Object.

DIS DODAG Information Solicitation.

EB Enhanced Beacon.

FDP Frame Delivery Probability.

FPU Floating-Point Unit.

FSM Finite State Machine.

ICMP Internet Control Message Protocol.

IE Information Element.

IP Internet Protocol.

IPv6 Internet Protocol versione 6.

JP Join Proxy.

KA Keep Alive.

LR-WPAN Low-Rate Wireless Personal Area Networks.

MP2P Multipoint-to-Point.

MSF Minimal Scheduling Function.

MTU Maximum Transmission Unit.

NACK Negative ACK.

P2MP Point-to-Multipoint.

PDR Packet Delivery Ratio.

PRIL Proactive Reduction of Idle Listening.

REST REpresentational State Transfer.

RPL IPv6 Routing Protocol for Low-Power and Lossy Networks.

RSSI Received Signal Strength Indication.

SF Scheduling Function.

TDMA Time Division Multiple Access.

TSCH Time Slotted Channel Hopping.

WSAN Wireless Sensor and Attuator Network.

WSN Wireless Sensor Network.

Termini

ν-exchange

Processo di scambio coerente dell'informazione aggiuntiva ν relativa alla nuova configurazione di rete.

bundle

Insieme di celle che sono state allocate per la comunicazione in direzione di uno specifico nodo.

channel hopping

Tecnica che prevede una variabilità nella selezione del canale logico su cui trasmettere e orientata a ridurre gli effetti prolungati di un'interferenza esterna nei confronti di uno o di un gruppo di canali logici.

channel offset

Indice che identifica la specifica cella all'interno dell'insieme di celle simultanee schedulate o schedulabili.

double listening

Intervallo temporale associato ad una doppia fase di ascolto, da parte del nodo ricevitore, sulla cella principale e sulla cella secondaria.

drift

Desincronizzazione relativa del clock del timer rispetto al comportamento standard.

guard time

Intervallo di tempo sufficientemente lungo che consente di ovviare al drift relativo dei timer dei nodi.

idle listening

Fenomeno inevitabile associato all'esecuzione di routine di ascolto da parte del nodo ricevitore in corrispondenza di assenza di informazione trasmessa dal nodo trasmettitore.

IEEE 802.11

Insieme di standard di trasmissione su reti WLAN su cui si basano le tecnologie Wi-Fi.

IEEE 802.15.4

Standardardizzazione del livello PHY (fisico) e MAC (collegamento dati) nel contesto di reti *wireless* a corto raggio e basso *rate* di trasmissione dei dati (LR-WPAN).

nodo foglia

o leaf node. Dispositivo che ricopre il ruolo di sensore o attuatore all'interno del sistema. Tipicamente trasmette e/o riceve piccole quantità di informazione e si pone in uno stato di *sleep* per la maggior parte del proprio ciclo di vita.

OpenWSN

Progetto open-source che fornisce gli strumenti per implementare applicativi che operano in un contesto di WSN e per interfacciarsi ad essi attraverso la rete IPv6.

overprovisioning

Tecnica che consiste nel sovrallocare risorse di rete in modo da poter soddisfare improvvisi picchi di traffico.

slot offset

Indice che identifica lo specifico gruppo di celle all'interno dello slotframe secondo la dimensione temporale.

slotframe

Struttura bidimensionale composta da un determinato numero di timeslot caratterizzati dalla possibilità di comunicare simultaneamente su diversi canali logici.

three way handshake

Protocollo che consente di instaurare una connessione attraverso lo scambio di tre appositi messaggi.

timeslot

Intervallo temporale durante il quale vengono effettuate le operazioni di ricezione e trasmissione dei frame.

TSCHpredictor

Simulatore Python ad eventi discreti che modella una semplice rete TSCH e consente di implementare e valutare nuovi algoritmi.

Bibliografia

[1] Douglas Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. Technical report, META Group, February 2001.

[2] Stefano Scanzio, Lukasz Wisniewski, and Piotr Gaj. Heterogeneous and dependable networks in industry – A survey. *Computers in Industry*, 125:103388, 2021.

[3] Antoine Bagula. Applications Of Wireless Sensor Networks. `http://wireless.ictp.it/wp-content/uploads/2012/02/WSN-Applications.pdf`. [Online; accessed 10-February-2022].

[4] Daniele Puccinelli and Martin Haenggi. Wireless sensor networks: applications and challenges of ubiquitous sensing. *IEEE Circuits and systems magazine*, 5(3):19–31, 2005.

[5] Marco Zennaro, Bjorn Pehrson, and Antoine Bagula. Wireless Sensor Networks: a great opportunity for researchers in Developing Countries. In *Proceedings of WCITD2008 Conference, Pretoria, South Africa*, volume 67, 2008.

[6] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, page 88–97, New York, NY, USA, 2002. Association for Computing Machinery.

[7] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. *SIGOPS Oper. Syst. Rev.*, 36(5):96–107, oct 2002.

[8] Paritosh Padhy, Kirk Martinez, Alistair Riddoch, Royan Ong, and J. Hart. Glacial Environment Monitoring using Sensor Networks. 01 2005.

[9] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pages 108–120, 2005.

[10] Rania Gomaa, Ihab Adly, Karam Sharshar, Ahmed Safwat, and Hani Ragai. ZigBee wireless sensor network for radiation monitoring at nuclear facilities. In *6th Joint IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 1–4, 2013.

[11] Mr Pragadeswaran, Ms Madhumitha, and Dr Gopinath. Certain Investigations on Military Applications of Wireless Sensor Networks. *International Journal of Advanced Research in Science, Communication and Technology*, pages 14–19, 03 2021.

[12] Ala Khalifeh, Khalid Darabkh, Ahmad Khasawneh, Issa Alqaisieh, Mohammad Salameh, Ahmed AlAbdala, Shams Alrubaye, Anwar Alassaf, Samer Al-HajAli, Radi Al-Wardat, Novella Bartolini, Giancarlo Bongiovanni, and Kishore Rajendiran. Wireless Sensor Networks for Smart Cities: Network Design, Implementation and Performance Evaluation. *Electronics*, 10:218, 01 2021.

[13] Stefano Scanzio, Mohammad Ghazi Vakili, Gianluca Cena, Claudio Giovanni Demartini, Bartolomeo Montrucchio, Adriano Valenzano, and Claudio Zunino. Wireless Sensor Networks and TSCH: A Compromise Between Reliability, Power Consumption, and Latency. *IEEE Access*, 8:167042–167058, 2020.

[14] Gianluca Cena, Claudio G. Demartini, Mohammad Ghazi Vakili, Stefano Scanzio, Adriano Valenzano, and Claudio Zunino. Evaluating and Modeling IEEE 802.15.4 TSCH Resilience against Wi-Fi Interference in New-Generation Highly-Dependable Wireless Sensor Networks. *Ad Hoc Networks*, 106:102199, 2020.

[15] Thomas Watteyne, Ankur Mehta, and Kristofer Pister. Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense. pages 116–123, 10 2009.

[16] Xavier Vilajosana, Thomas Watteyne, Tengfei Chang, Mališa Vučinić, Simon Duquennoy, and Pascal Thubert. IETF 6TiSCH: A Tutorial. *IEEE Communications Surveys Tutorials*, 22(1):595–615, 2020.

[17] IEEE Standard for Low-Rate Wireless Networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, 2020.

[18] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez, Fabien Chraim, Kevin Weekly, Qin Wang, Steven Glaser, and Kris Pister. OpenWSN: a standards-based low-power wireless development environment. *Transactions on Emerging Telecommunications Technologies*, 23(5):480–493, 2012.

[19] Xenofon Fafoutis, Atis Elsts, George Oikonomou, Robert Piechocki, and Ian Craddock. Adaptive static scheduling in IEEE 802.15.4 TSCH networks. In *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pages 263–268, 2018.

[20] Gianluca Cena, Stefano Scanzio, Lucia Seno, Adriano Valenzano, and Claudio Zunino. Energy-Efficient Link Capacity Overprovisioning In Time Slotted Channel Hopping Networks. In *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, pages 1–8, 2020.

[21] J.R. Vig. Quartz Crystal Resonators and Oscillators - For Frequency Control and Timing Applications - A Tutorial, 10 2014.

[22] Tengfei Chang, Thomas Watteyne, Kris Pister, and Qin Wang. Adaptive synchronization in multi-hop TSCH networks. *Computer Networks*, 76:165–176, 2015.

[23] Giada Giorgi and Claudio Narduzzi. Performance Analysis of Kalman-Filter-Based Clock Synchronization in IEEE 1588 Networks. *IEEE Transactions on Instrumentation and Measurement*, 60(8):2902–2909, 2011.

[24] Maurizio Mongelli and Stefano Scanzio. Approximating Optimal Estimation of Time Offset Synchronization With Temperature Variations. *IEEE Transactions on Instrumentation and Measurement*, 63(12):2872–2881, 2014.

[25] Maurizio Mongelli and Stefano Scanzio. A neural approach to synchronization in wireless networks with heterogeneous sources of noise. *Ad Hoc Networks*, 49:1–16, 2016.

[26] Stefano Scanzio, Gianluca Cena, Adriano Valenzano, and Claudio Zunino. Energy Saving in TSCH Networks by Means of Proactive Reduction of Idle Listening. In Luigi Alfredo Grieco, Gennaro Boggia, Giuseppe Piro, Yaser Jararweh, and Claudia Campolo, editors, *Ad-Hoc, Mobile, and Wireless Networks*, pages 131–144, Cham, 2020. Springer International Publishing.

[27] Stefano Scanzio, Gianluca Cena, Lucia Seno, and Adriano Valenzano. Robustness and Optimization of PRIL Techniques for Energy Saving in TSCH Networks. In *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2021.

[28] Xavier Vilajosana, Kris Pister, and Thomas Watteyne. Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration. RFC 8180, May 2017.

[29] Tengfei Chang, Mališa Vučinić, Xavier Vilajosana, Simon Duquennoy, and Diego Roberto Dujovne. 6TiSCH Minimal Scheduling Function (MSF). RFC 9033, May 2021.

[30] Qin Wang, Xavier Vilajosana, and Thomas Watteyne. 6TiSCH Operation Sublayer (6top) Protocol (6P). RFC 8480, November 2018.

[31] Jonathan W. Hui and David E. Culler. Extending IP to Low-Power, Wireless Personal Area Networks. *IEEE Internet Computing*, 12(4):37–45, 2008.

[32] Tsvetko Tsvetkov and Alexander Klein. RPL: IPv6 routing protocol for low power and lossy networks. *Network*, 59:59–66, 2011.

[33] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.

[34] Peng Du and George Roussos. Adaptive channel hopping for wireless sensor networks. In *2011 International Conference on Selected Topics in Mobile and Wireless Networking (iCOST)*, pages 19–23, 2011.

[35] Rasool Tavakoli, Majid Nabi, Twan Basten, and Kees Goossens. Enhanced Time-Slotted Channel Hopping in WSNs Using Non-intrusive Channel-Quality Estimation. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems*, pages 217–225, 2015.

[36] Vasileios Kotsiou, Georgios Z. Papadopoulos, Periklis Chatzimisios, and Fabrice Tholeyre. Is local blacklisting relevant in slow channel hopping low-power wireless networks? In *2017 IEEE International Conference on Communications (ICC)*, pages 1–6, 2017.

[37] Vasileios Kotsiou, Georgios Z. Papadopoulos, P. Chatzimisios, and Fabrice Théoleyre. LABeL: Link-based Adaptive BLacklisting Technique for 6TiSCH Wireless Industrial Networks. *Proceedings of the 20th ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, 2017.

[38] Stefano Scanzio, Mohammad Ghazi Vakili, Gianluca Cena, Claudio Giovanni Demartini, Bartolomeo Montrucchio, Adriano Valenzano, and Claudio Zunino. Wireless Sensor Networks Dataset (TSCH a Compromise Between Reliability, Power Consumption, and Latency), 2020.

[39] J. Postel. Transmission Control Protocol. RFC 793, Internet Engineering Task Force, September 1981.

[40] Municio Esteban, Daneels Glenn, Malisa Vucinic, Steven Latré, Jeroen Famaey, Yasuyuki Tanaka, Keoma Brun-Laguna, Xavier Vilajosana, Kazushi Muraoka, and Thomas Watteyne. Simulating 6TiSCH Networks. *Transactions on emerging telecommunications technologies*, August 2018.

[41] Atis Elsts. TSCH-Sim: Scaling Up Simulations of TSCH and 6TiSCH Networks. *Sensors*, 20(19), 2020.

[42] Xavier Vilajosana, Qin Wang, Fabien Chraim, Thomas Watteyne, Tengfei Chang, and Kristofer S. J. Pister. A Realistic Energy Consumption Model for TSCH Networks. *IEEE Sensors Journal*, 14(2):482–489, 2014.

[43] Gianluca Cena, Stefano Scanzio, and Adriano Valenzano. Enabling Listening Suspension in the Time Slotted Channel Hopping Protocol. In *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, pages 19–26, 2021.