UNIVERSITÀ DI PADOVA

FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA

IL PROTOCOLLO CoAP SU SISTEMI WiFi A BASSO CONSUMO

Laureando: Marco Nicolosi

Relatore: Prof. Lorenzo Vangelista

Corso di Laurea magistrale in Ingegneria delle Telecomunicazioni

Anno accademico 2013/2014

Padova, 14 aprile 2014

Indice

Capitolo 1

Introduzione

1.1 Internet delle cose

Il termine *Internet delle cose* è un termine che si riferisce ad una evoluzione del concetto di rete Internet agli oggetti concreti comuni che diventano univocamente identificabili in tale struttura di rappresentazione virtuale.

Una visione di questo tipo prevede la possibilità di connettere oggetti fisici estranei al mondo computazionale e renderli disponibili con modalità che vanno oltre gli scenari di rete tradizionali. Si parla più precisamente di computazione ubiqua [5] e pervasiva [10], ovvero di integrazione della computazione in sempre più aspetti del mondo concreto e di arricchimento di capacità computazionali per categorie sempre maggiori di oggetti.

Le comunicazioni Macchina-Macchina (M2M) si riferiscono a qualsiasi tipo di tecnologia che renda possibile lo scambio di informazioni tra dispositivi e l'esecuzione di operazioni senza l'intervento di umani.

Ci si aspetta una crescita notevole dei sistemi di comunicazione M2M nelle prossime decadi soprattutto nelle seguenti aree: controllo e gestione di utenze (elettricità, gas, riscaldamento e acqua), sistemi di trasporto intelligente (ITS), sorveglianza, automazione domestica ed industriale.

1.1.1 Architettura REST

REST è l'acronimo di REpresentational State Transfer, termine coniato da Roy Fielding nella sua tesi di dottorato [4]. L'architettura REST definisce la nozione di *risorsa* e mira a modellizzare tutte le interazioni client/server come uno scambio di *rappresentazioni di risorse*. L'obiettivo è quello di realizzare una infrastruttura di gestione delle risorse remote tramite alcune semplici funzioni di accesso e interazione come quelle di HTTP: PUT, POST, GET, DELETE.

Il concetto di risorsa nel Web è spesso confuso col concetto di rappresentazione della risorsa. Una risorsa è un concetto astratto che viene assegnato ad un identificatore universale URI che può essere usato nel Web. L'accesso ad un identificatore URI tuttavia non potrà mai fornire come risposta la risorsa ma solo una sua rappresentazione. Ciascuna risorsa può avere varie rappresentazioni tra cui la più popolare è il formato a documento (HTML o PDF) e il formato della lingua parlata. HTTP possiede delle funzionalità che consentono ai client di richiedere delle particolari rappresentazioni e di negoziare il contenuto.

Per risorse fisiche come tag RFID è chiaro che il Web non potrà mai garantire l'accesso alla risorsa ma solo alla sua rappresentazione ovvero alla rappresentazione dello stato corrente della risorsa al momento della richiesta.

L'accesso alle rappresentazioni delle risorse può venire automatizzato nell'ottica dell'Internet delle cose e lo sviluppo di protocolli efficienti che garantiscano questo a dispositivi con risorse computazionali ed energetiche limitate è un aspetto importante che verrà analizzato nel prossimo capitolo dove verrà presentato il protocollo CoAP.

Capitolo 2

Il Protocollo CoAP

Uno degli obiettivi principali del protocollo CoAP è realizzare un protocollo WEB adatto alle esigenze di dispositivi con risorse limitate in termini computazionali ed energetiche. Il modo con cui si vuole realizzare questo protocollo non consiste in una semplice compressione del protocollo HTTP, quanto piuttosto nella implementazione di un sottoinsieme delle funzionalità offerte dalla architettura REST in comune con HTTP ottimizzando queste nell'ottica delle applicazioni di comunicazione M2M.

Il lavoro di standardizzazione del protocollo è opera del Constrained RESTful Environments (CoRE) Working Group della Internet Engineering Task Force (IETF) [11].

2.1 Caratteristiche

Le principali caratteristiche che il protocollo CoAP presenta sono:

- protocollo web per nodi di rete con risorse limitate adatto a comunicazioni macchina-macchina;
- trasporto su protocollo a datagramma come UDP con affidabilità opzionale;
- scambio asincrono di messaggi;
- basso overhead e bassa complessità di parsing dell'header;
- supporto a risorse URI e ad informazioni content-type del payload;
- realizzazione in modo semplice d'intermediari (proxy);
- possibilità di memorizzare in cache risposte per ridurre tempi di risposta ed occupazione di banda;
- traducibilità nel protocollo privo di stati HTTP con possibilità di realizzare proxy per garantire a nodi HTTP l'accesso a risorse CoAP e viceversa.

2.2 Formato del messaggio

L'interazione tra nodi CoAP avviene in modo simile al modello client/server del protocollo HTTP. Tuttavia la natura delle interazioni macchina-macchina che avvengono tra dispositivi remoti nell'Internet delle cose suggerisce una implementazione del protocollo CoAP in cui ogni nodo agisca sia da client che da server. Un tale nodo viene detto *end-point*. Una richiesta CoAP è equivalente ad una richiesta HTTP: essa è inviata da un client ad un server per richiedere a questo l'esecuzione di una azione (tramite un codice di metodo) su una risorsa (identificata tramite URI). Il

server invia poi al client originario una risposta contenente un codice di risposta ed una eventuale rappresentazione della risorsa richiesta.

A differenza di HTTP, il protocollo CoAP realizza questi scambi di richieste/risposte in maniera asincrona su un protocollo di trasporto a datagramma come UDP. Il modo con cui si realizza questo è tramite uno livello protocollare di messaggi che possano supportare una affidabilità opzionale tramite algoritmo di backoff esponenziale. I messaggi CoAP possono essere di 4 tipi: Confermabile (CON), Non confermabile (NON), Acknowledgement (ACK) e Reset (RST). I codici di metodo e di risposta inclusi in alcuni di questi messaggi specificano che si tratta di una richiesta o di una risposta.

Si può pensare al protocollo CoAP come ad un livello composto da due sottolivelli (figura 2.1):

- un sottolivello di *messaggistica* che si occupa della gestione dello scambio di messaggi che come detto prima è asincrono e vincolato ad UDP;
- un sottolivello delle interazioni richiesta/risposta che utilizza i codici di Metodo e di Risposta per elaborare la richiesta o risposta.

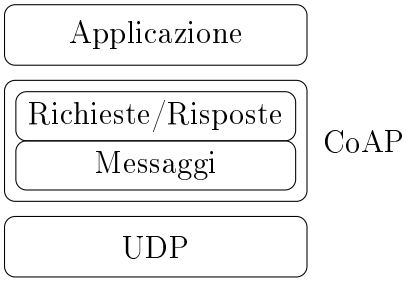


Figura 2.1: Struttura a livelli del protocollo CoAP.

Formato del messaggio L'informazione trasportata da un messaggio CoAP è codificata in formato binario nei seguenti blocchi illustrati in figura 2.2:

- il primo elemento è un *header* di lunghezza fissa pari a 4 byte;
- segue il campo contenente il valore del *token* di lunghezza variabile tra 0 ed 8 byte;
- dopo il *token* vi è una eventuale sequenza di zero o più opzioni CoAP codificate in formato *type-length-value*;
- infine, dopo la sequenza di opzioni, può seguire un eventuale *payload* che occupa la parte rimanente del datagramma. Se tale payload è presente e di dimensione non nulla, esso viene preceduto da un marker lungo 1 byte formato da bit tutti unitari (0xFF) che sta ad indicare la fine del campo delle opzioni e l'inizio del campo Payload.

I campi dell'*header* sono:

Ver: *Versione*, intero non negativo di 2 bit. Indica la versione del protocollo CoAP implementata nel nodo. Poichè la standardizzazione del protocollo non è ancora stata ultimata, tale versione viene impostata ad 1 riservando numeri successivi a versioni future.

T: *Tipo*, intero non negativo di 2 bit. Indica se il messaggio è di tipo *Confermabile* (0), *Non-Confermabile* (1), *Acknowledgement* (2) o *Reset* (3). Il significato di tali tipi di messaggio è indicato nelle sezioni 2.3.1 e 2.3.2.

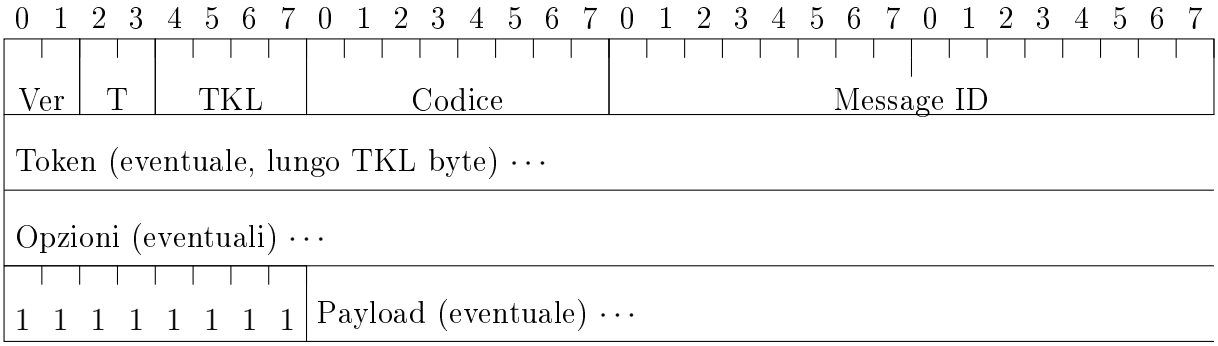


Figura 2.2: Formato del messaggio CoAP.

TKL: *Token Length*, intero non negativo di 4 bit. Indica la lunghezza del campo token che come detto è variabile da 0 a 8 byte. Il suo valore serve a correlare richieste e risposte come indicato in sezione 2.4.4. Lunghezze che vanno da 9 a 15 byte sono riservate, non vanno inviate e se ricevute devono essere interpretate come errore del messaggio del formato.

Codice: intero non negativo di 8 bit. Indica se il messaggio è una richiesta (codici 1-31), una risposta (codici 64-191) o è vuoto (codice 0). Tutti gli altri codici sono riservati. Nel caso di una richiesta il campo *Codice* ne indica il Metodo (cfr. sezione 2.4.1) mentre nel caso di una risposta tale campo indica il Codice di Risposta (sezione 2.4.2).

Message ID: intero non negativo di 16 bit con ordinamento dei byte *network byte order* (big-endian). E' utilizzato per la rivelazione di messaggi duplicati e per associare messaggi di tipo Acknowledgement/Reset a messaggi di tipo Confermabile/Non-Confermabile secondo le regole di generazione ed associazione descritte nella sezione 2.4.4.

2.2.1 Formato delle opzioni

Dopo l'header e l'eventuale token segue una eventuale sequenza di zero o più opzioni. Il tipo di opzione è individuato da un *Numero Opzione* (ON) all'interno di una lista di possibili *Opzioni CoAP* indicate in tabella 2.5. All'interno di ciascuna opzione è specificato il Numero Opzione ON, la lunghezza del valore dell'opzione (OL) e il valore stesso dell'opzione.

Invece di indicare direttamente il Numero Opzione ON, la lista dei campi opzioni deve essere ordinata in base al Numero Opzione e si utilizza una codifica delta tra di esse: il Numero Opzione ON_n di ciascuna opzione n-esima è calcolato come somma del suo Delta Opzione OD_n e del Numero Opzione ON_{n-1} dell'opzione precedente

$$\mathrm{ON}_n = \mathrm{OD}_n + \mathrm{ON}_{n-1}$$

dove il Delta Opzione, indicato nei primi 4 bit di ciascun campo opzione, è appunto definito come la differenza tra i Numeri Opzione di due opzioni adiacenti

$$\mathrm{OD}_n = \mathrm{ON}_n - \sum_{j=0}^{n-1} \mathrm{OD}_j$$

Per calcolare il Delta Opzione OD_1 della prima opzione ($n = 1$) di un messaggio si assume che la precedente opzione (non presente nel messaggio) abbia Numero Opzione $\mathrm{ON}_0 \triangleq 0$, ovvero

$$\mathrm{OD}_n = \mathrm{ON}_n - \sum_{j=1}^{n-1} \mathrm{OD}_j\,, \quad \text{per } n \geq 1$$

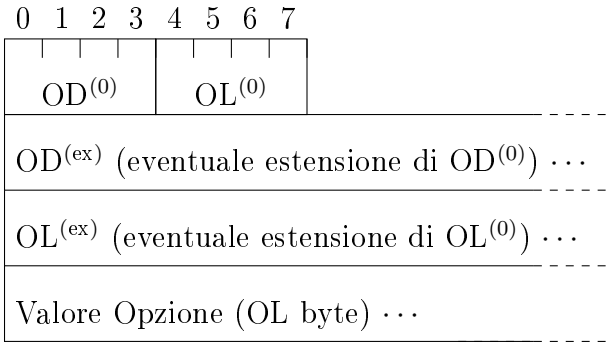


Figura 2.3: Formato del campo Opzione CoAP.

Più occorrenze dello stesso tipo di opzione possono essere incluse semplicemente utilizzando un Delta Opzione nullo tra di esse.

La struttura di un campo opzione è formato da tre sottocampi (figura 2.3):

Delta Opzione ($\mathrm{OD}^{(0)}$): intero non negativo di 4 bit. Un valore di $\mathrm{OD}^{(0)} \in [0, 12]$ esprime il Delta Opzione OD:

$$\mathrm{OD} = \mathrm{OD}^{(0)} \quad \text{per } \mathrm{OD}^{(0)} \in [0, 12]\,.$$

I rimanenti tre valori del sottocampo $\mathrm{OD}^{(0)}$ sono riservati per indicare la presenza di costrutti speciali:

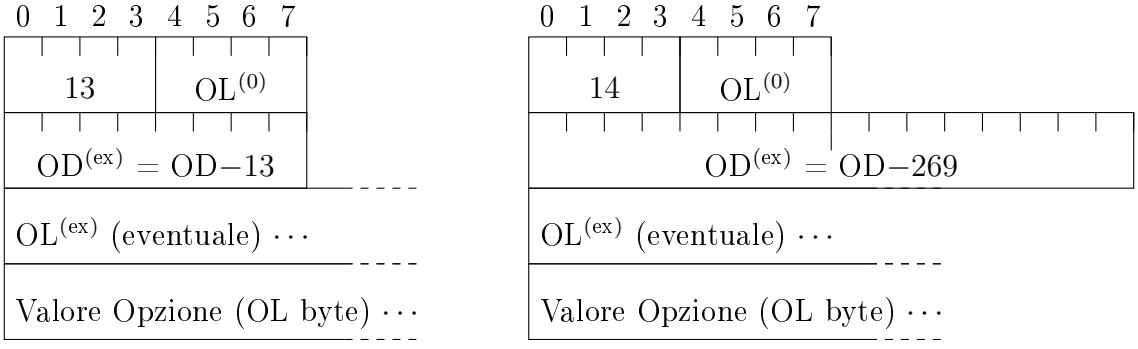
13: un intero non negativo di 8 bit $\mathrm{OD}^{(\mathrm{ex})}$ segue il byte iniziale del campo opzione ed indica il Delta Opzione OD meno 13 (figura 2.4(a)):

$$\mathrm{OD} = \mathrm{OD}^{(\mathrm{ex})} + 13 \quad \text{per } \mathrm{OD}^{(0)} = 13\,;$$

14: un intero non negativo di 16 bit $\mathrm{OD}^{(\mathrm{ex})}$ con ordinamento dei byte *network byte order* (big-endian) segue il byte iniziale del campo opzione ed indica il Delta Opzione OD meno 269 (figura 2.4(b)):

$$\mathrm{OD} = \mathrm{OD}^{(\mathrm{ex})} + 269 \quad \text{per } \mathrm{OD}^{(0)} = 14\,;$$

15: valore riservato per il marker del Payload.



(a) $13 \leq \mathrm{OD} \leq 2^8 - 1 + 13 = 268$ (b) $269 \leq \mathrm{OD} \leq 2^{16} - 1 + 269 = 65804$

Figura 2.4: Formato del campo Delta Opzione estesa.

Lunghezza Opzione ($\mathrm{OL}^{(0)}$): intero non negativo di 4 bit. Un valore di $\mathrm{OL}^{(0)} \in [0, 12]$ esprime la Lunghezza Opzione OL:

$$\mathrm{OL} = \mathrm{OL}^{(0)} \quad \text{per } \mathrm{OL}^{(0)} \in [0, 12]\,.$$

I rimanenti tre valori del sottocampo $OL^{(0)}$ sono riservati per indicare la presenza di costrutti speciali:

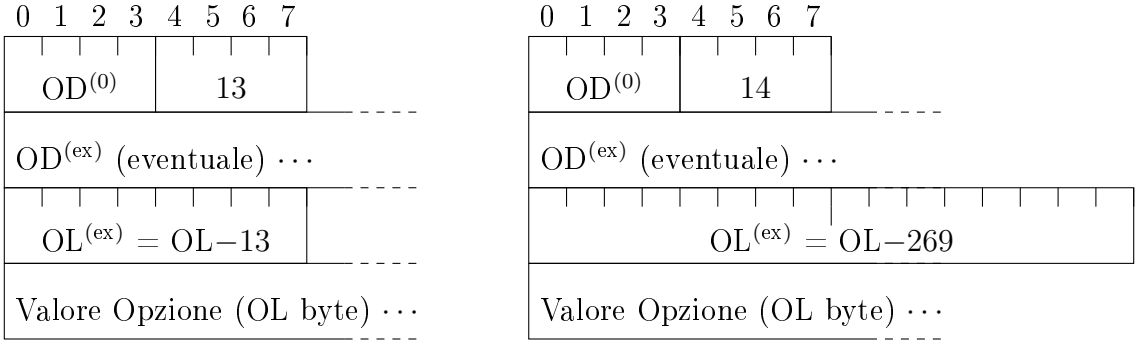
13: un intero non negativo di 8 bit $OL^{(ex)}$ precede il sottocampo Valore Opzione ed indica la Lunghezza dell'Opzione OL meno 13 (figura 2.5(a)):

$$OL = OL^{(ex)} + 13 \quad \text{per } OL^{(0)} = 13\,;$$

14: un intero non negativo di 16 bit $OL^{(ex)}$ con ordinamento dei byte *network byte order* (big-endian) precede il sottocampo Valore Opzione ed indica la Lunghezza dell'Opzione OL meno 269 (figura 2.5(b)):

$$OL = OL^{(ex)} + 269 \quad \text{per } OL^{(0)} = 14\,;$$

15: valore riservato per uso futuro. Nel caso tale valore sia presente nel campo $OL^{(0)}$ di un messaggio, deve essere considerato come errore di formato del messaggio.



(a) $13 \leq OL \leq 2^8 - 1 + 13 = 268$ (b) $269 \leq OL \leq 2^{16} - 1 + 269 = 65804$

Figura 2.5: Formato del campo Lunghezza Opzione estesa.

Valore Opzione: sequenza di OL byte che rappresenta il valore dell'opzione.

2.3 Trasmissione dei messaggi

Come indicato in precedenza, il protocollo CoAP è vincolato ad utilizzare un protocollo di trasporto non affidabile come UDP. I messaggi possono pertanto giungere al destinatario non in ordine, arrivare duplicati o andare persi senza che ciò venga notificato. Per ovviare a questo inconveniente il protocollo CoAP supporta un sottolivello protocollare di messaggi leggero che garantisca affidabilità opzionale senza ricreare tutto l'insieme di funzioni di un servizio di trasporto come TCP. Le principali caratteristiche sono:

- affidabilità per messaggi confermabili tramite semplici ritrasmissioni con backoff esponenziale;
- rilevamento di duplicati per messaggi sia Confermabili che Non-Confermabili.

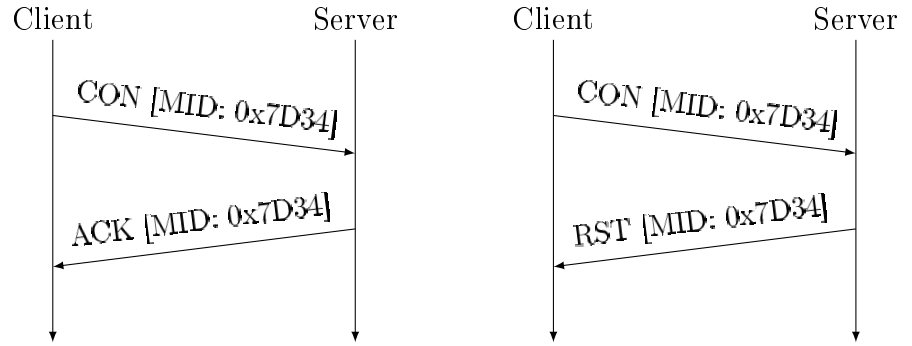
2.3.1 Trasmissione affidabile

Una trasmissione affidabile di un messaggio avviene specificando nell'header che il messaggio è di tipo Confermabile. Un messaggio Confermabile trasporta sempre o una richiesta od una risposta

tranne nel caso in cui sia usato solamente per provocare l'invio da parte del destinatario di un messaggio di tipo Reset, nel qual caso tale messaggio è vuoto[1].

Ogni destinatario, alla ricezione di un messaggio Confermabile, deve:

- confermare con un messaggio di riscontro di tipo ACK la avvenuta ricezione del messaggio elaborato ed interpretato correttamente (figura 2.6(a));
- scartare il messaggio rispondendo con un messaggio di tipo Reset (figura 2.6(b)) nel caso manchi parte del contesto per poterlo elaborare e/o interpretare correttamente (ad es. se il messaggio è vuoto, il codice messaggio appartiene ad una classe riservata, si rileva un errore di formato o il destinatario a seguito di riavvio ha perso delle informazioni di stato necessarie per la corretta elaborazione del messaggio).



(a) Ricezione di un messaggio ACK (b) Ricezione di un messaggio RST

Figura 2.6: Trasmissione affidabile di messaggi CoAP.

Ogni messaggio di Acknowledgement deve avere lo stesso Message ID del del messaggio Confermabile di cui deve confermare la ricezione e deve o trasportare una risposta (codice 64-191) oppure essere vuoto (codice 0) (si veda la sezione 2.4.2). Ogni messaggio di Reset deve avere lo stesso Message ID del del messaggio Confermabile di cui deve confermare la ricezione e deve essere vuoto.

La ricezione di messaggi di tipo Acknowledgement o Reset non deve essere notificata con l'invio di altri messaggi di tipo Acknowledgement o Reset. Qualora si riceva un messaggio di Acknowledgement o di Reset che non possa essere interpretato correttamente (ad es. se il codice di un ACK è una richiesta o di una classe riservata oppure se un RST non è vuoto) esso va scartato semplicemente ignorandolo.

Ritrasmissione Ogni messaggio Confermabile viene ritrasmesso dal mittente dopo intervalli di tempo crescenti esponenzialmente fino a quando non si riceve un ACK, RST oppure si raggiunge un numero massimo stabilito di possibili ritrasmissioni.

I parametri che controllano la ritrasmissione di un messaggio CON in attesa dell'ACK/RST sono un timer T_i^{rtx} (per ogni trasmissione i-esima) e un contatore di ritrasmissioni N_{rtx} impiegati per realizzare un algoritmo di backoff esponenziale binario:

- all'invio di un nuovo messaggio CON il timer iniziale T_0^{rtx} viene posto ad un valore casuale compreso tra un valore minimo di attesa T_{ACK} e $\alpha_{\text{ACK}} T_{\text{ACK}}$ (dove $\alpha_{\text{ACK}} > 1$ è un fattore noto indicato in tabella 2.1) mentre il contatore di ritrasmissioni N_{rtx} è posto a 0,

$$T_0^{\text{rtx}} \in [T_{\text{ACK}},\ \alpha_{\text{ACK}} T_{\text{ACK}}]\,, \qquad N_{\text{rtx}} = 0\,; \tag{2.1}$$

[1] Si tratta della procedura di ping CoAP, usata ad esempio per controllare lo stato di attività di un host.

- prima del raggiungimento del timeout del timer $T_i^{\text{rtx}}, i \geq 0$
  - se si riceve un messaggio di ACK, la trasmissione del messaggio CON è considerata avvenuta con successo;
  - se si riceve un messaggio di RST, la trasmissione del messaggio CON è considerata fallita;
- al raggiungimento del timeout del timer $T_i^{\text{rtx}}, i \geq 0$
  - se $N_{\text{rtx}} < N_{\text{rtx}}^{\text{MAX}}$ si ritrasmette il messaggio, si incrementa di 1 il contatore N_{rtx} e si imposta un nuovo timer $T_{i+1}^{\text{rtx}} = 2T_i^{\text{rtx}}$;
  - se $N_{\text{rtx}} = N_{\text{rtx}}^{\text{MAX}}$ la trasmissione del messaggio CON è considerata fallita.

Parametro	Descrizione	Valore predefinito
T_{ACK}	Tempo minimo di attesa iniziale dell'ACK	2 s
α_{ACK}	Fattore per il massimo tempo di attesa iniziale dell'ACK	1.5
$N_{\text{rtx}}^{\text{MAX}}$	Numero massimo di ritrasmissioni	4

Tabella 2.1: Parametri di trasmissione e ritrasmissione CoAP

Chiaramente tale algoritmo può essere terminato anche prima che venga raggiunta una delle condizioni appena indicate. Può infatti ad esempio succedere che una richiesta CON di un client produca come risposta dal server un ACK vuoto in attesa della risposta CON Separate (sezione 2.4.2). Se il client dopo aver inviato la richiesta CON, in attesa dell'ACK, riceve soltanto una risposta CON è chiaro che l'ACK vuoto è andato perso e che si è ricevuta una risposta Separate. Risulta pertanto inutile procedere alla ritrasmissione della richiesta CON iniziale e ragionevole terminare l'algoritmo di backoff esponenziale.

2.3.2 Trasmissione non affidabile

I messaggi che non richiedano una trasmissione affidabile come ad esempio la comunicazione periodica di una lettura di un sensore dove è sufficiente anche una conoscenza non immediata dello stato della risorsa monitorata, possono essere trasmessi senza richiedere riscontro da parte del destinatario specificando nell'header che il messaggio è di tipo *Non-Confermabile* (NON).

Un messaggio di tipo Non-Confermabile trasporta sempre o una richiesta od una risposta e non può essere vuoto.

Ogni destinatario alla ricezione di un messaggio Non-Confermabile non deve inviare un messaggio di riscontro al mittente (figura 2.7). Qualora il destinatario si accorga di non essere in grado di elaborare e/o interpretare correttamente il messaggio[2] deve scartare il messaggio ignorandolo semplicemente ed eventualmente inviando un messaggio di RST corrispondente.

Poiché il mittente di una richiesta Non-Confermabile *non* può avere riscontro della avvenuta ricezione del messaggio da parte del destinatario, può scegliere di inviare copie multiple dello stesso messaggio entro l'intervallo di tempo $T_{\text{TX,SPAN}}^{\text{MAX}}$ (definito in sezione 2.3.4) oppure la rete può duplicare il pacchetto in transito. Tali copie multiple devono avere lo stesso Message ID del

[2]Sono inclusi ad esempio i casi in cui il messaggio sia vuoto, il codice messaggio appartenga ad una classe riservata, si rilevi un errore di formato o il destinatario a seguito di riavvio abbia perso delle informazioni di stato necessarie per la corretta elaborazione del messaggio.

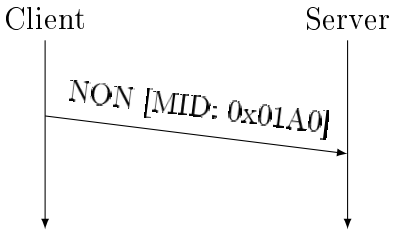


Figura 2.7: Trasmissione non affidabile di messaggi CoAP.

messaggio originario di modo che il ricevitore alla ricezione di più messaggi con lo stesso Message ID elabori il messaggio una ed una sola volta.

In tabella 2.2 sono riassunti i possibili utilizzi dei quattro tipi di messaggio CoAP. L'asterisco indica che la combinazione non è utilizzata nel normale scambio di messaggi ma solo in modalità ping per ottenere dal destinatario l'invio di un messaggio di tipo RST.

	CON	**NON**	**ACK**	**RST**
Richiesta	✓	✓	–	–
Risposta	✓	✓	–	–
Vuoto	*	–	✓	✓

Tabella 2.2: Uso dei Tipi di messaggio CoAP

2.3.3 Associazione dei messaggi

Un riscontro di tipo ACK o RST è associato ad un messaggio CON o NON attraverso il suo Message ID presente nell'header del messaggio. Esso viene generato dal mittente di una richiesta CON/NON in modo che lo stesso Message ID non sia riutilizzato all'interno della finestra temporale T_{EXCHANGE} (sezione 2.3.4) per comunicare con lo stesso host. Tale Message ID viene quindi riportato dal destinatario nell'header del messaggio di riscontro ACK/RST per consentire al mittente la corretta associazione con il messaggio CON/NON corrispondente.

Il metodo più semplice per generare un diverso Message ID ogni volta che si comunica con lo stesso host all'interno di T_{EXCHANGE} consiste nel memorizzare staticamente una variabile per valore del Message ID e cambiarla ogniqualvolta venga inviata una nuova richiesta CON/NON indipendentemente dall'host di destinazione. Esistono 2^{16} possibili Message ID.

Come detto in precedenza esiste la possibilità che un destinatario riceva più copie dello stesso messaggio CON/NON (avente cioè stesso message ID, stesso indirizzo e stessa porta di origine) all'interno di T_{EXCHANGE} o T_{NON} (sezione 2.3.4). Questo può accadere ad esempio quando dei riscontri ACK/RST non giungano a destinazione o arrivino dopo il timeout di ritrasmissione del mittente oppure quando un host decide di inviare più copie dello stesso messaggio NON. In tale situazione l'host che riceve tali duplicati deve

- inviare un riscontro ACK identico per ogni messaggio CON ricevuto all'interno di T_{EXCHANGE} ed elaborare ogni richiesta/risposta del messaggio *una sola volta* scartando i duplicati;
- ignorare ogni messaggio NON duplicato ricevuto all'interno di T_{NON} ed elaborare ogni richiesta/risposta del messaggio *una sola volta* scartando i duplicati.

2.3.4 Parametri di trasmissione

I valori di T_{ACK}, α_{ACK} e $N_{\text{rtx}}^{\text{MAX}}$ definiti nella sezione 2.3.1 determinano la temporizzazione dell'algoritmo di ritrasmissione e di conseguenza anche per quanto tempo è necessario mantenere memorizzate certe informazioni.

Si definiscano i seguenti intervalli temporali:

- $T^{\text{MAX}}_{\text{TX,SPAN}}$ sia il tempo massimo che intercorre tra la prima trasmissione di un messaggio CON all'istante prima della sua ultima ritrasmissione:

$$T^{\text{MAX}}_{\text{TX,SPAN}} = \sum_{i=0}^{N^{\text{MAX}}_{\text{rtx}}-1} T_{\text{ACK}}\alpha_{\text{ACK}}2^i = T_{\text{ACK}}\alpha_{\text{ACK}}\left(2^{N^{\text{MAX}}_{\text{rtx}}}-1\right)$$

- $T^{\text{MAX}}_{\text{TX,WAIT}}$ sia il tempo massimo che intercorre tra la prima trasmissione di un messaggio CON all'istante in cui il mittente smette di attendere la ricezione di un riscontro ACK o RST:

$$T^{\text{MAX}}_{\text{TX,WAIT}} = \sum_{i=0}^{N^{\text{MAX}}_{\text{rtx}}} T_{\text{ACK}}\alpha_{\text{ACK}}2^i = T_{\text{ACK}}\alpha_{\text{ACK}}\left(2^{N^{\text{MAX}}_{\text{rtx}}+1}-1\right)$$

- $T^{\text{MAX}}_{\text{LATENCY}}$ sia il tempo massimo che si suppone intercorra tra l'inizio della trasmissione di un datagramma e la sua completa ricezione; tale parametro viene posto in [11] pari a 100 secondi in linea con la scelta del Maximum Segment Lifetime (MSL[3]) per il protocollo TCP;

- T_{PROC} sia il tempo massimo che un host impiega per generare il messaggio di riscontro ACK dalla ricezione del messaggio CON; se si suppone che l'host proceda all'invio dell'ACK prima che scada il timeout di ritrasmissione per il mittente, esso è pari al massimo a T_{ACK};

- $T^{\text{MAX}}_{\text{RTT}}$ sia il massimo tempo di round-trip:

$$T^{\text{MAX}}_{\text{RTT}} = 2\,T^{\text{MAX}}_{\text{LATENCY}} + T_{\text{PROC}}$$

Dagli intervalli temporali appena definiti possono essere derivati dei parametri operativi relativi a ciascuna specifica implementazione del protocollo CoAP:

- T_{EXCHANGE} è definito come il tempo che intercorre tra l'invio di un messaggio CON e l'istante in cui il mittente di tale messaggio è sicuro di non poter più ricevere riscontro:

$$\begin{aligned} T_{\text{EXCHANGE}} &= T^{\text{MAX}}_{\text{TX,SPAN}} + T^{\text{MAX}}_{\text{RTT}} \\ &= T^{\text{MAX}}_{\text{TX,SPAN}} + 2\,T^{\text{MAX}}_{\text{LATENCY}} + T_{\text{PROC}}\,; \end{aligned}$$

  si noti che non è stato preso in considerazione il parametro $T^{\text{MAX}}_{\text{TX,WAIT}}$ in quanto si è supposto che il valore $T^{\text{MAX}}_{\text{LATENCY}}$, scelto come caso pessimo, sia maggiore dell'intervallo di attesa dell'ultimo timeout di ritrasmissione $T^{\text{MAX}}_{\text{TX,WAIT}} - T^{\text{MAX}}_{\text{TX,SPAN}} = T_{\text{ACK}}\alpha_{\text{ACK}}2^{N^{\text{MAX}}_{\text{rtx}}}$; questa considerazione impone anche che il numero massimo di ritrasmissioni $N^{\text{MAX}}_{\text{rtx}}$ non possa essere posto oltre un valore limite tale per cui $T^{\text{MAX}}_{\text{TX,WAIT}} - T^{\text{MAX}}_{\text{TX,SPAN}} \leq T^{\text{MAX}}_{\text{LATENCY}}$, ovvero

$$N^{\text{MAX}}_{\text{rtx}} \leq \log_2 \frac{T^{\text{MAX}}_{\text{LATENCY}}}{T_{\text{ACK}}\alpha_{\text{ACK}}}\,;$$

- T_{NON} è definito come il tempo che intercorre tra l'invio di un messaggio NON e l'istante in cui il mittente può riutilizzare il Message ID di tale messaggio; nel caso di trasmissioni non multiple tale valore è pari a $T^{\text{MAX}}_{\text{LATENCY}}$; per trasmissioni multiple, come ad esempio applicazioni multicast, è ragionevole aspettarsi duplicati in un intervallo di tempo pari a $T^{\text{MAX}}_{\text{TX,SPAN}}$ e pertanto si può porre per sicurezza

$$T_{\text{NON}} = T^{\text{MAX}}_{\text{TX,SPAN}} + T^{\text{MAX}}_{\text{LATENCY}}\,.$$

Qualora si voglia utilizzare un unico timer per gestire il riuso di Message ID generati per messaggi CON e NON si può utilizzare il massimo tra T_{EXCHANGE} e T_{NON}, ovvero T_{EXCHANGE}.

Con i valori standard dei parametri della tabella 2.1 si possono derivare i seguenti valori dei parametri definiti in questa sezione, riportati in tabella 2.3

[3]Il Maximum Segment Lifetime viene definito in [8] come il tempo di vita massimo atteso per un segmento TCP all'interno della rete e posto arbitrariamente pari a 2 minuti.

Parametro	Descrizione	Valore predefinito
$T^{\text{MAX}}_{\text{TX,SPAN}}$	Tempo massimo tra la prima trasmissione di un messaggio CON e la sua ultima ritrasmissione	45 s
$T^{\text{MAX}}_{\text{TX,WAIT}}$	Tempo massimo tra la prima trasmissione di un messaggio CON e la fine dell'attesa di un riscontro	93 s
$T^{\text{MAX}}_{\text{LATENCY}}$	Tempo massimo tra l'inizio della trasmissione di un datagramma e la sua completa ricezione	100
T_{PROC}	Tempo massimo di generazione di riscontro ACK dalla ricezione di un messaggio CON	2 s
$T^{\text{MAX}}_{\text{RTT}}$	Tempo massimo di round-trip	202 s
T_{EXCHANGE}	Tempo tra l'invio di un messaggio CON e l'istante in cui il mittente è sicuro di non poter più ricevere riscontro	247 s
T_{NON}	Tempo necessario per poter riutilizzare il Message ID di un messaggio CON	145 s

Tabella 2.3: Parametri temporali di trasmissione CoAP

2.4 Richieste e risposte

La comunicazione tra host CoAP avviene secondo un modello di tipo richiesta/risposta simile a quello HTTP: un client invia una o più richieste ad un server che elabora la richiesta ed invia una risposta. A differenza di HTTP le richieste e risposte non vengono inviate attraverso una connessione stabilita in precedenza ma sono scambiate in modo asincrono attraverso il sottolivello di messaggi CoAP.

2.4.1 Richieste

Una richiesta CoAP è formata da un *metodo* da applicare ad una risorsa, l'identificatore URI di tale risorsa, un eventuale payload con indicatore del formato (Content-Format, sezione 2.4.5) ed eventuali metadati aggiuntivi.

Un messaggio confermabile o non confermabile di richiesta viene creato specificando nel campo Codice dell'header il codice di Metodo della richiesta (codici 1-31) assieme ad altre informazioni aggiuntive incluse nel messaggio.

Un host che riceva una richiesta avente codice di metodo non riconosciuto o non supportato deve inviare una risposta piggy-backed con codice di risposta 4.05 (Method Not Allowed).

Metodi

I metodi supportati dal protocollo sono un sottoinsieme dei metodi di richiesta del protocollo HTTP: GET, POST, PUT, DELETE (tabella 2.4).

GET Il metodo GET richiede di ottenere una rappresentazione dell'informazione corrispondente alla risorsa identificata dall'opzione URI inclusa nel messaggio.

Può includere una opzione Accept che specifica il formato (Content-Format) preferito per la risposta, oppure una opzione ETag che richiede l'invio di una risposta che confermi l'eventuale validità della risposta corrispondente memorizzata o l'invio della rappresentazione della risorsa nel caso la validità non sia stata confermata.

I codici di risposta possono essere 2.05 (Content) oppure 2.04 (Valid) nel caso la risposta salvata corrispondente all'ETag indicato venga confermata valida.

Il metodo GET è sicuro[4] ed idempotente[5].

POST Il metodo POST richiede che la rappresentazione della risorsa identificata dal campo URI venga elaborata tramite una funzione eseguita dal server dove tale risorsa risiede. Tale funzione viene stabilita dal server stesso ed è dipendente dalla risorsa indicata. Solitamente il risultato di tale operazione è l'aggiornamento della risorsa indicata oppure la creazione di una nuova risorsa se questa non è presente nel server.

Il codice di risposta è 2.01 (Created) nel caso sia stata creata una nuova risorsa (nel qual caso si possono includere una o più opzioni Location-Path e/o Location-Query relativi alla nuova risorsa), 2.04 (Changed) nel caso sia stata modificata una risorsa già presente senza crearne una nuova oppure 2.02 (Deleted) nel caso la risorsa indicata sia stata cancellata.

Il metodo POST non è né sicuro né idempotente.

PUT Il metodo PUT richiede che la risorsa identificata dal campo URI indicato nel messaggio venga aggiornata oppure creata con la rappresentazione inclusa nella richiesta. Il formato di rappresentazione può essere specificato includendo una opzione Content-Format.

Se la risorsa indicata dall'URI esiste, la rappresentazione inclusa nel messaggio è da considerare una versione modificata della risorsa stessa e si deve rispondere con un codice 2.04 (Changed) qualora la modifica sia avvenuta con successo. Se la risorsa indicata non esiste nel server, questo può crearne una nuova identificata dall'URI indicato rispondendo con codice 2.01 (Created). Qualora invece la risorsa indicata non possa essere né modificata né creata, il server deve rispondere con un codice di errore appropriato. Ulteriori restrizioni al metodo PUT possono essere imposte tramite le opzioni If-Match o If-None-Match (sezione 2.4.5).

Il metodo PUT non è sicuro ma è idempotente.

DELETE Il metodo DELETE richiede che la risorsa identificata dal campo URI indicato nel messaggio venga eliminata. Se l'operazione di eliminazione della risorsa avviene con successo o la risorsa indicata non esisteva prima della richiesta, il messaggio di risposta deve avere codice 2.02 (Deleted).

Il metodo DELETE non è sicuro ma è idempotente.

Codice	Nome	Sicuro	Idempotente
0.01	GET	✓	✓
0.02	POST	–	–
0.03	PUT	–	✓
0.04	DELETE	–	✓

Tabella 2.4: Metodi di richiesta CoAP[6].

2.4.2 Risposte

Dopo aver ricevuto ed interpretato una richiesta, un server risponde con un messaggio di risposta CoAP che è associato alla richiesta originaria tramite il token generato dal client e incluso nella messaggio di richiesta.

[4] Un metodo è detto *sicuro* quando esso ha solo la funzione di ottenimento di una rappresentazione di una risorsa (di cui pertanto non modifica lo stato).

[5] Un metodo è detto *idempotente* quando l'effetto generato sul server da più richieste con tale metodo è lo stesso provocato da una sola richiesta.

[6] Il codice di metodo è indicato secondo la notazione `classe.dettaglio` utilizzata per i codici di risposta (sezione 2.4.2).

Analogamente al Codice di Stato nel protocollo HTTP, una risposta CoAP è contraddistinta da un Codice Risposta (codici 64-191), presente nel campo Codice dell'header del messaggio, che indica l'esito del tentativo di comprendere e soddisfare la richiesta ricevuta.

Codici di risposta

Gli 8 bit che formano il Codice Risposta sono suddivisi secondo la notazione `classe.dettaglio` (figura 2.8):

- i 3 bit più significativi specificano la classe della risposta;
- i rimanenti 5 bit meno significativi specificano un dettaglio aggiuntivo all'interno della classe di risposta.

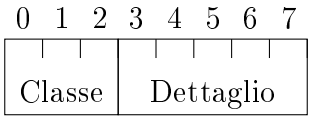


Figura 2.8: Struttura del Codice Risposta CoAP.

Esistono 3 classi di risposta:

2 - Successo: la richiesta è stata ricevuta, interpretata e accettata;

4 - Errore del client: la richiesta contiene errori di sintassi o non può essere soddisfatta;

5 - Errore del server: il server non è riuscito a soddisfare una richiesta accettata come valida.

Qualora un host riceva un codice di risposta non supportato o non riconosciuto all'interno della classe 4 (Errore del client) o 5 (Errore del server) essi vanno trattati come codici di risposta equivalenti al generico codice di risposta di quella classe (4.00 o 5.00 rispettivamente). Se si riceve un codice di risposta non supportato o non riconosciuto all'interno della classe 2 (Successo), poiché non esiste un generico codice di risposta 2.00 all'interno di tale classe che indichi esito positivo della richiesta, l'host deve interpretare tale risposta semplicemente come un riscontro della riuscita della richiesta senza ulteriori dettagli aggiuntivi.

I codici di risposta sono descritti nel dettagio qui di seguito.

Classe 2 - Successo

2.01 Created In risposta solo a richieste aventi metodo POST o PUT, indica (analogamente al codice HTTP 201 Created) l'esito positivo del tentativo di creazione della risorsa indicata nella richiesta.

Può includere un eventuale payload che è la rappresentazione della risorsa creata e opzioni come Location-Path e/o Location-Query i cui valori indicano la posizione in cui è stata creata la risorsa. Se tali opzioni non sono presenti la risorsa è stata creata nella posizione identificata dall'URI della richiesta.

La risposta 2.01 Created non è memorizzabile in cache e un host che riceva una tale risposta deve etichettare una eventuale precedente risposta in cache come *not fresh*.

2.02 Deleted In risposta solo a richieste aventi metodo POST o PUT, indica (analogamente al codice HTTP 201 Created) l'esito positivo del tentativo di creazione della risorsa indicata nella richiesta.

2.03 Valid In risposta ad una richiesta contenente l'opzione ETag, serve ad indicare che la risposta identificata dall'entity-tag presente nella opzione ETag inclusa è valida. Una risposta 2.03 non deve contenere payload e deve invece includere una opzione ETag.

Un client che riceva una risposta 2.03 Valid deve aggiornare il campo Max-Age della risposta salvata in cache con il campo Max-Age incluso nella opzione della risposta ricevuta.

2.04 Changed Utilizzata in risposta ad una richiesta POST o PUT, come il codice HTTP 204 No Content indica che la risorsa indicata dal campo URI è stata modificata.

2.05 Content Analoga alla risposta HTTP 200 OK ma utilizzata solo in risposta ad una richiesta GET. Il payload incluso nella risposta è la rappresentazione della risorsa richiesta.

Classe 4 - Client Error

4.00 Bad Request Come la risposta HTTP 400 Bad Request.

4.01 Unauthorized Il client non è autorizzato ad effettuare la azione richiesta e non deve ritrasmettere una tale richiesta prima di aver eseguito l'autenticazione al server.

4.02 Bad Option La richiesta non è stata compresa dal server a causa di una o più opzioni non riconosciute o contenenti errori di sintassi.

4.03 Forbidden Come la risposta HTTP 403 Forbidden.

4.04 Not Found Come la risposta HTTP 404 Not Found.

4.05 Not Allowed Come la risposta HTTP 405 Not Allowed senza il campo header Allow.

4.06 Not Acceptable Come la risposta HTTP 406 Not Acceptable senza entità di risposta.

4.12 Precondition Failed Come la risposta HTTP 412 Precondition Failed.

4.13 Request Entity Too Large Il server non può elaborare una richiesta della dimensione della richiesta e può includere una opzione Size1 per indicare il limite della dimensione dell'entità di richiesta che il server è in grado di accettare.

4.15 Unsupported Content-Format Come la risposta HTTP 415 Unsupported Media Type.

Classe 5 - Server Error

5.00 Internal Server Error Come la risposta HTTP 500 Internal Server Error.

5.01 Not Implemented Come la risposta HTTP 501 Not Implemented.

5.02 Bad Gateway Come la risposta HTTP 502 Bad Gateway.

5.03 Service Unavailable Come la risposta HTTP 503 Service Unavailable utilizzando l'opzione Max-Age al posto del campo header HTTP Retry-After per indicare il numero di secondi dopo cui riprovare ad inviare la richiesta.

5.04 Gateway Timeout Come la risposta HTTP 504 Gateway Timeout.

5.04 Proxying Not Supported Il server non è in grado di operare in modalità proxy per l'URI specificato nell'opzione Proxy-Uri o utilizzando lo schema dell'opzione Proxy-Scheme.

Risposta Piggy-backed

Se la risposta ad una richiesta di tipo CON è disponibile immediatamente, il server può inviarla direttamente nel messaggio di riscontro (ACK) della richiesta CON appena ricevuta (figura 2.9(a)). In questo modo si elimina la necessità di dover trasmettere la risposta in un secondo messaggio distinto dal riscontro della richiesta ricevuta. Questo tipo di risposta viene detto *piggy-backed* e non necessita di un riscontro in quanto qualora non giungesse a destinazione, il client provvederebbe a ritrasmettere la richiesta per al più $N_{\text{rtx}}^{\text{MAX}}$ volte fino alla eventuale ricezione del riscontro. In figura 2.9 sono illustrati due casi di risposta *piggy-backed* in cui vengono persi dei messaggi: nella figura 2.9(b) viene perso il messaggio di richiesta mandato dal client e la sua ritrasmissione genera la risposta *piggy-backed* che giunge a destinazione; nella figura 2.9(c) il primo messaggio di risposta *piggy-backed* viene perso ed allo scadere del timeout T_0^{rtx} il client procede alla ritrasmissione del messaggio di richiesta che giungerà a destinazione.

Risposta Separate

Può accadere che la risposta ad un messaggio di richiesta di tipo CON non sia immediatamente disponibile per varie ragioni. Ad esempio il server può impiegare per l'ottenimento della rappresentazione della risorsa richiesta un tempo maggiore del tempo di attesa del timeout di ritrasmissione del client, nel qual caso quest'ultimo procederà alla ritrasmissione del messaggio di richiesta.

In questi casi è opportuno che nel server venga avviato un timer di riscontro in contemporanea all'inizio del processo di ottenimento della rappresentazione della risorsa. Allo scadere di questo timer il server procede all'invio del riscontro ACK della richiesta CON ricevuta. Se la rappresentazione della risorsa viene ottenuta prima del timeout allora si procede al suo invio *piggy-backed* nel riscontro (stesso Message ID e stesso token). Se invece allo scadere del timeout il server non è riuscito ad ottenere la rappresentazione della risorsa il server procede all'invio del riscontro della richiesta CON ricevuta come ACK vuoto (che verrà ritrasmesso nel caso di ricezione di duplicati della stessa richiesta).

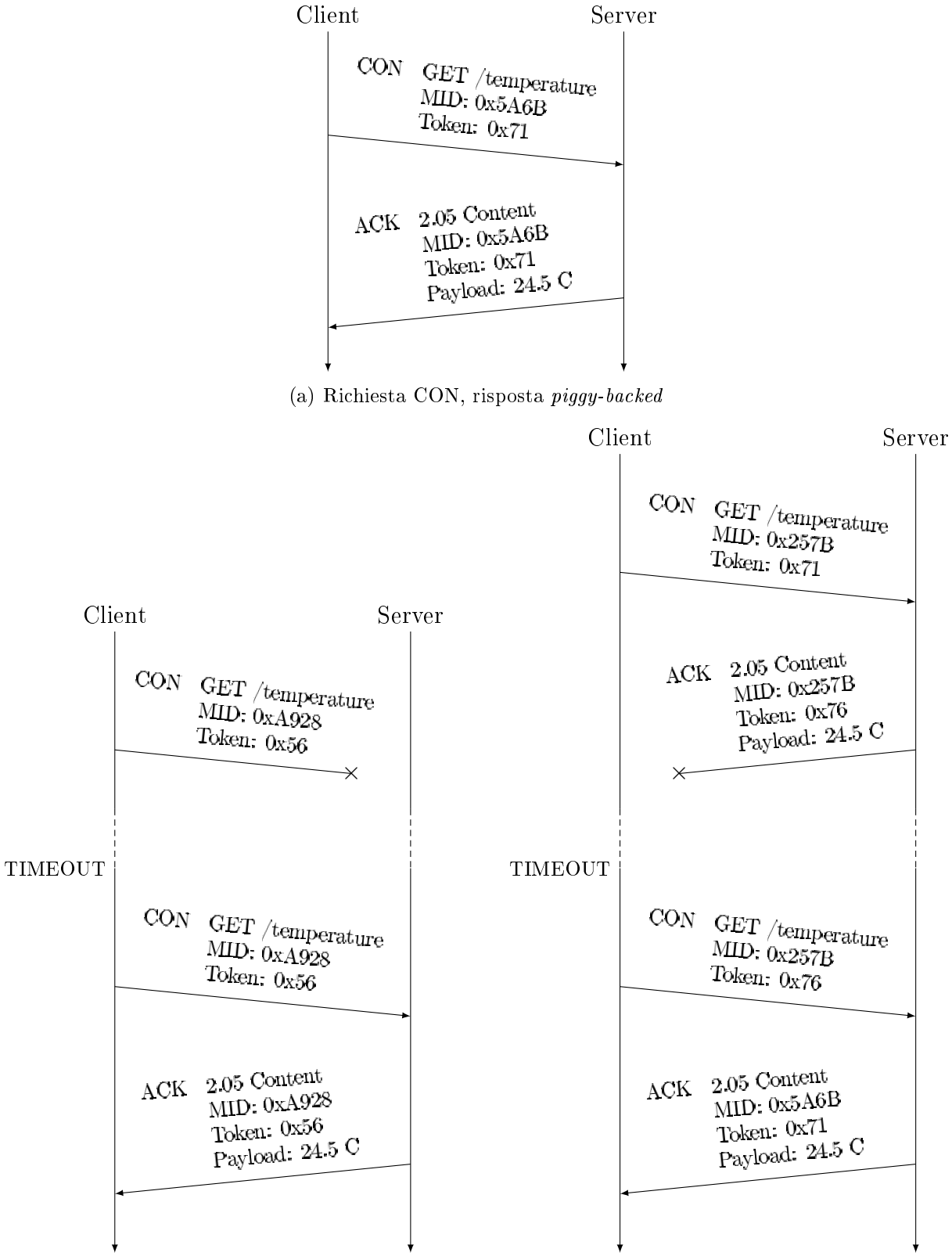
Appena il server ottiene la rappresentazione della risorsa richiesta, la invia al client in un nuovo messaggio CON avente quindi un diverso Message ID ma lo stesso token della richiesta originaria (si veda la sezione successiva). Tale tipo di risposta è detta *separate*. Con l'invio della risposta *separate* in un messaggio di tipo CON il server avvia a sua volta un timer di ritrasmissione allo scadere del quale in assenza di riscontro procederà al reinvio della risposta secondo l'algoritmo descritto in sezione 2.3.1. Alla ricezione della risposta CON il client deve a sua volta inviare un riscontro ACK vuoto al server (un riscontro che non trasporti né una richiesta né una risposta).

In figura 2.10 sono illustrate due risposte *separate* a due richieste di tipo CON. In particolare nella figura 2.10(a) è mostrato il caso in cui il primo ACK vuoto del server alla richiesta del client non giunge a destinazione: il client deve essere consapevole del fatto che qualora l'ACK vada perso può ricevere una risposta CON alla sua richiesta senza riscontro oppure può riceverlo dopo, non essendo garantito l'ordine di recapito dei pacchetti dal protocollo UDP.

2.4.3 Token

Ogni richiesta inviata da client trasporta un token di lunghezza variabile da 0 a 8 byte che il server deve riportare senza modifica nel messaggio di risposta. Si noti che anche un token vuoto (di 0 byte) è un token valido.

Il token come accennato nella sezione 2.2 è utilizzato per associare una risposta alla richiesta originaria scegliendola tra le varie richieste che possono essere ancora in attesa di risposta nel client. Si noti che un client può scegliere di utilizzare lo stesso token per inviare diverse richieste ad uno stesso host a patto che il socket locale associato a ciascuna richiesta sia diverso, ovvero che la porta sorgente sia diversa. La regola che va rispettata per garantire differenziazione tra le varie risposte è quella di utilizzare token diversi per ciascuna coppia (host sorgente, host di



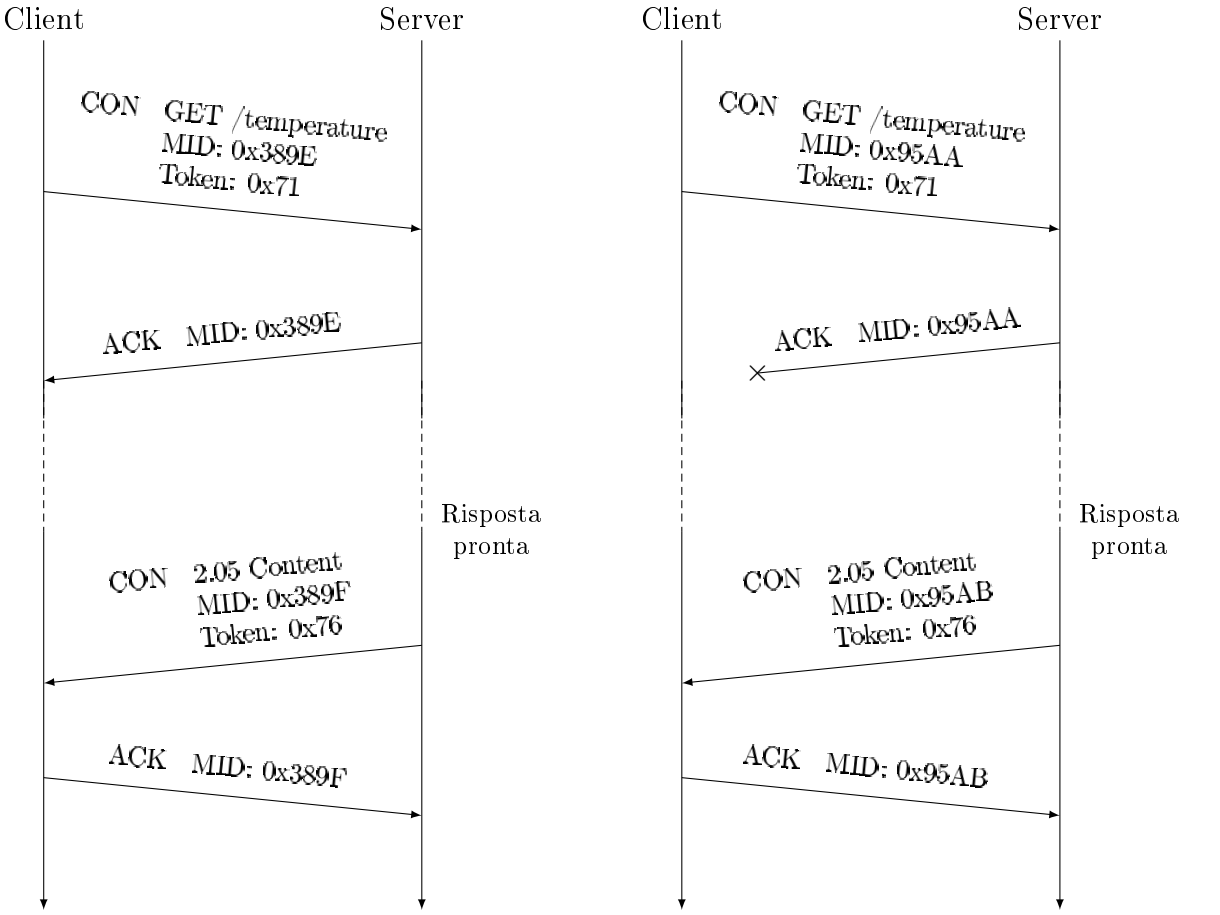
(a) Richiesta CON, risposta *piggy-backed*

(b) Richiesta CON persa e ritrasmessa, risposta *piggy-backed*

(c) Richiesta CON ritrasmessa, risposta *piggy-backed* persa e ritrasmessa

Figura 2.9: Richieste CoAP di tipo CON con risposte *piggy-backed*.

destinazione) dove con host si intende un entità all'interno di un nodo caratterizzata da indirizzo IP e porta (quindi un socket).



(a) Richiesta CON, risposta *separate*

(b) Richiesta CON ritrasmessa, risposta *piggy-backed* persa e ritrasmessa

Figura 2.10: Richieste CoAP di tipo CON con risposte *separate*.

2.4.4 Associazione richieste/risposte

La regola con cui un nodo CoAP associa una risposta ricevuta alla corrispondente richiesta è la seguente:

- l'indirizzo remoto IP e la porta UDP del nodo che ha inviato la risposta devono essere gli stessi della richiesta originaria;
- in una risposta *piggy-backed* sia il Message ID che il token presente nell'ACK di risposta devono coincidere con quelli della richiesta CON originaria;
- in una risposta *separate* solo il token della risposta CON originaria deve coincidere con quello della richiesta CON originaria;

2.4.5 Opzioni

Le varie opzioni che è possibile includere in un messaggio CoAP sono riportate in tabella 2.5 assieme ai rispettivi numeri di opzione Il dettaglio di ciascuna opzione si trova in [11].

2.4.6 URI

Il protocollo CoAP utilizza una sintassi per gli identificatori universali di risorse URI che segue lo standard [1]. Un URI CoAP può avere uno *scheme* di tipo `coap` oppure di tipo `coaps` nel caso

Numero	Nome Opzione
1	If-Match
3	Uri-Host
4	ETag
5	If-None-Match
7	Uri-Port
8	Location-Path
11	Uri-Path
12	Content-Format
14	Max-Age
15	Uri-Query
17	Accept
20	Location-Query
35	Proxy-Uri
39	Proxy-Scheme
60	Size1

Tabella 2.5: Numeri Opzione CoAP

vengano utilizzato il protocollo DTLS per lo scambio sicuro di messaggi. Verrà riportato qui solo lo schema `coap`.

Il formato dello schema di un URI `coap` è

```
"coap://" <host> [":" <port>] <path-abempty> ["?" <query>]
```

Il campo `host` può contenere un indirizzo IP letterale o IPv4 che permette di raggiungere il server all'indirizzo specificato oppure può essere un nome registrato, ovvero un identificatore indiretto che va convertito in un indirizzo IP tramite un servizio di risoluzione dei nomi quale ad esempio DNS per conoscere l'indirizzo dell'host. Non può essere vuoto ed un URI contenente un campo `host` vuoto deve essere considerato non valido.

Il campo `port` indica la porta UDP su cui il server è in ascolto. Se tale campo non è presente si assume che il server sia in ascolto sulla porta standard 5683.

Il campo `path` consiste in una sequenza di segmenti separati dal carattere `/` ed identifica univocamente la risorsa all'interno dell'host e porta indicati.

Il campo `query` indica una parametrizzazione della risorsa richiesta e consiste in una sequenza di argomenti separati dal carattere `&`.

2.5 Traduzione CoAP/HTTP

Il protocollo CoAP supporta un sottoinsieme limitato delle funzionalità HTTP pertanto la traduzione di un messaggio CoAP in un messaggio HTTP è semplice da effettuare. Le ragioni che possono spingere ad usare dei proxy di traduzione tra i due protocolli sono molteplici, ad esempio per rendere compatibili tra loro nodi che utilizzano protocolli differenti.

Le traduzioni che rendono possibili le interazioni e l'accesso alle risorse tra nodi con protocolli differenti sono di due tipi:

Proxying CoAP-HTTP Permette ai client CoAP di accedere a risorse presenti su server HTTP attraverso un intermediario. Una richiesta di questo tipo è avviata includendo una opzione Proxy-Uri o Proxy-Scheme contenente un URI http o https in un messaggio di richiesta inviato ad un proxy CoAP-HTTP.

Proxying CoAP-HTTP Permette ai client HTTP di accedere a risorse presenti su server CoAP attraverso un intermediario. Una richiesta di questo tipo è avviata specificando un URI di tipo coap o coaps nella linea di richiesta di un messaggio di richiesta HTTP inviato ad proxy CoAP-HTTP.

La modalità con cui vengono implementate queste traduzioni è fuori dallo scopo di questo scritto e può essere trovato in [11].

2.6 Confronto CoAP/HTTP

Si illustrano qui di seguito le principali differenze tra il protocollo CoAP ed HTTP analizzando separatamente lo strato applicazione (messaggi CoAP e messaggi HTTP) e quello di trasporto (UDP e TCP).

2.6.1 Livello applicazione

Il protocollo HTTP così come definito in [3] è un protocollo senza stato[7] *stateless* nato per definire come i client Web richiedono risorse Web dai server creato da Fielding nella versione 1.1 in base all'architettura REST.

A seconda che venga trasportata una richiesta od una risposta i formati del messaggio HTTP sono quelli di figura 2.11 o 2.12. Ciascun campo è codificato in maniera testuale ASCII e separato da caratteri `Space` (SP, ASCII 0x20), `Carriage Return` (CR, ASCII 0x0D) o `Line Feed` (LF, ASCII 0x0A).

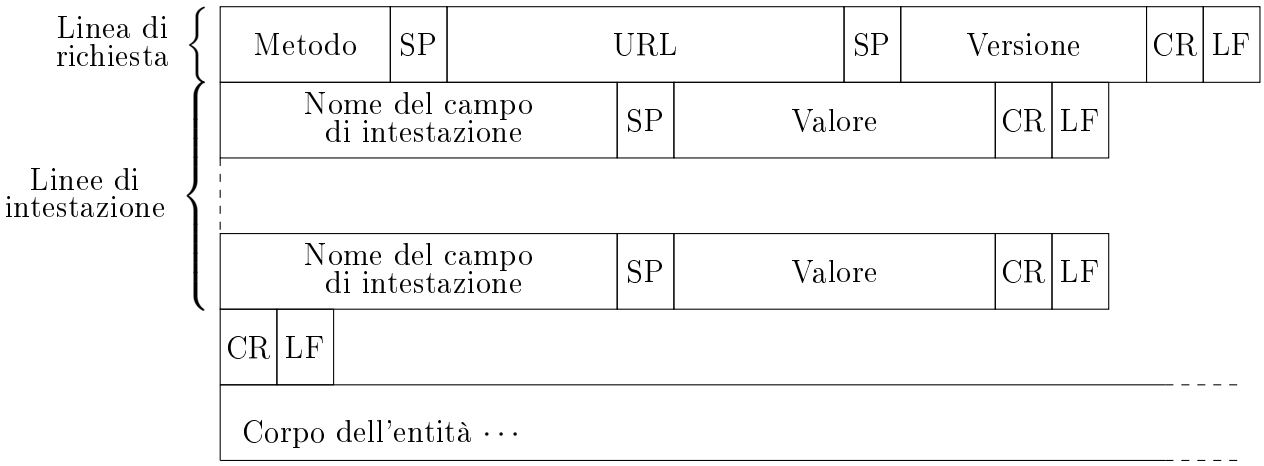


Figura 2.11: Formato di un messaggio di richiesta HTTP.

Una prima differenza che si nota subito è il formato testuale (ASCII) dei campi che compongono l'header del messaggio HTTP (linea di intestazione e linee di richiesta). Questo aspetto rende il protocollo HTTP poco adatto a nodi con risorse energetiche limitate per i quali invece CoAP risulta più efficiente nel trasportare messaggi di richiesta/risposta analoghi. Ad esempio il metodo di richiesta è codificato nel messaggio HTTP come sequenza di caratteri che può essere lunga dai 3 (GET) ai 7 (CONNECT) byte mentre in una richiesta CoAP la dimensione del campo Codice è di appena 1 byte. Analogamente nel messaggio di risposta HTTP il codice di stato e il messaggio di stato possono raggiungere dimensione di diversi byte, specialmente il messaggio di stato che risulta una ridondante descrizione in linguaggio umano del codice di stato e può esser lungo anche 31 byte

[7] Ad eccezione dei cookie di sessione un server HTTP risponde alle richieste che giungono da diversi client senza immagazzinare alcuna informazione di stato relativa al client.

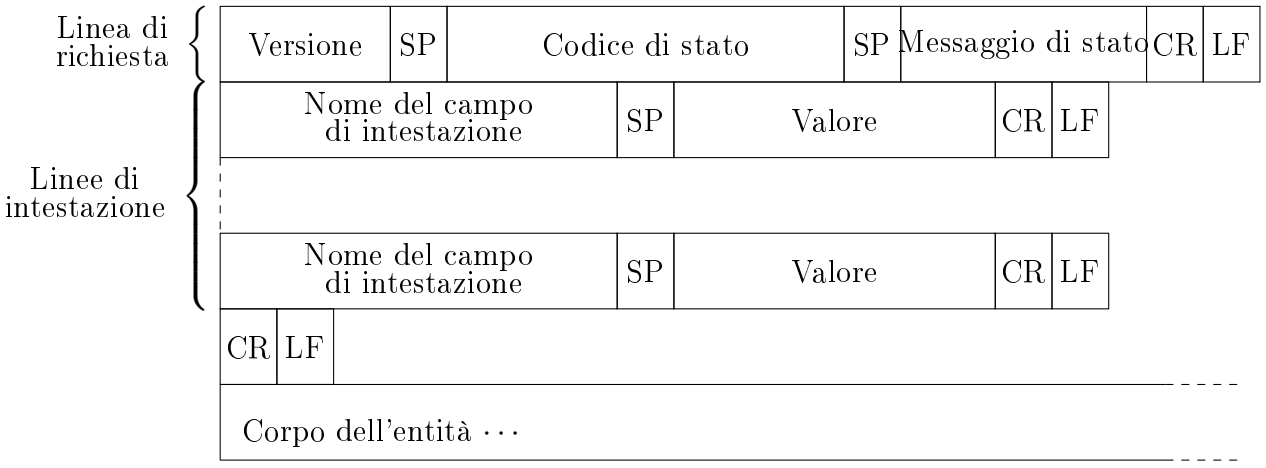


Figura 2.12: Formato di un messaggio di risposta HTTP.

(PROXY AUTHENTICATION REQUIRED); nel protocollo CoAP codice di stato e messaggio di stato risultano accorpati e codificati nel campo Codice occupando appena 1 byte.

L'utilizzo nel protocollo CoAP del Numero di Opzione e della codifica delta progressiva consente di ridurre ad $1 \div 5$ i byte necessari per codificare il tipo di opzione che in HTTP sono ancora riportati come sequenza di caratteri ASCII nelle linee di intestazione, arrivando ad occupare uno spazio dell'ordine delle decine di byte per ciascu nome del campo di intestazione.

HTTP può impiegare sia la connessione non persistente che quella persistente. La prima utilizza una diversa connessione TCP per il trasferimento di ogni oggetto con notevole dispendio di banda, energia. Con la connessione persistente il server lascia aperta la connessione TCP dopo aver spedito la risorsa richiesta e permette alle successive richieste e riposte di utilizzare la stessa connessione.

Il protocollo CoAP è nato per le esigenze di nodi con limitate risorse e esigenze di comunicazione poco frequenti che rischiano di diventare troppo distanti nel tempo per mantenere aperta la connessione TCP. Ecco perchè il livello di affidabilità dello scambio di messaggi CoAP non è affidato alla connessione TCP ma al sottolivello della logica CoAP che non prevede connessione ma semplicemente l'invio della risposta che da sola trasporta anche il riscontro della ricezione, o al massimo di una risposta separata dal riscontro. Se sono necessarie ritrasmissioni, HTTP si affida a TCP che implementa riscontri cumulativi delle ricezioni di segmenti. Tuttavia per lo scambio di datagrammi/segmenti di pochi byte questo tipo di riscontro non mostra differenze rispetto al riscontro di un intero datagramma come nel caso CoAP.

2.6.2 UDP/TCP

I protocolli dello stato di trasporto su cui si appoggiano CoAP ed HTTP possiedono caratteristiche differenti: il primo sfrutta un protocollo di trasporto inaffidabile e privo di connessione, UDP, mentre il secondo utilizza un protocollo che garantisce un servizio affidabile orientato alla connessione, TCP. In particolare, come si vedrà in dettaglio qui di seguito, il protocollo UDP fornisce soltanto due servizi minimali del livello di trasporto: l'estensione del servizio di spedizione di pacchetti tra due host fornito da IP alla spedizione di pacchetti tra due processi all'interno dei due host (multiplexing/demultiplexing tramite socket) e il controllo di errore (checksum). Il protocollo TCP invece, oltre a questi, offre servizi addizionali come ad esempio il trasferimento affidabile dei dati (spedizione corretta ed in ordine dei segmenti), i riscontri di consegna, i numeri di sequenza, il controllo di flusso ed il controllo della congestione.

UDP

Il protocollo UDP [7] è stato pensato per offrire le funzioni minime previste dallo strato di trasporto. Come si può vedere dalla struttura del suo header in figura 2.13 esso aggiunge soltanto quattro campi, ciascuno di lunghezza 2 byte, all'header ip:

- i numeri delle porte sorgente e di destinazione, per permettere all'host di destinazione di passare il datagramma al corretto processo (socket) attivo;
- la lunghezza del datagramma UDP, intestazione compresa, espressa in byte;
- la somma di controllo (*checksum*) utilizzata per effettuare il rilevamento di eventuali errori introdotti[8] nel datagramma UDP durante la trasmissione.

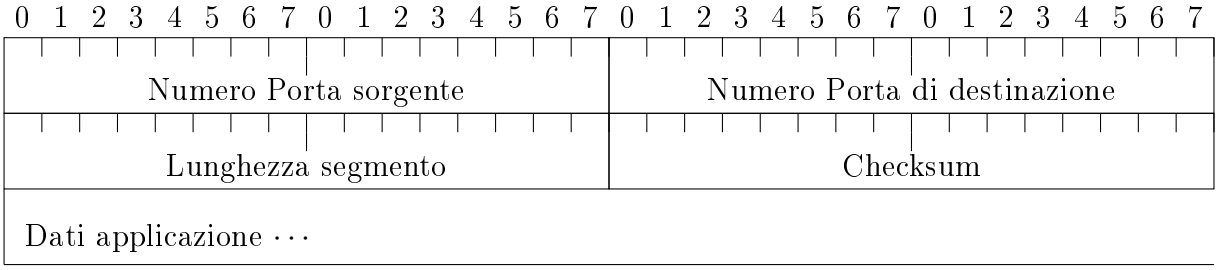


Figura 2.13: Formato del datagramma UDP.

Le funzioni minime offerte da UDP si rivelano particolarmente adatte ad applicazioni con risorse limitate, come nel caso del protocollo CoAP. In particolare UDP è preferibile a TCP per le seguenti ragioni:

Basso overhead d'intestazione. L'header UDP come già detto ha una dimensione di appena 8 byte contro i 20 byte richiesti dall'intestazione TCP.

Assenza di connessione. A differenza del protocollo TCP che richiede l'impostazione della connessione con l'host di destinazione mediante handshake a tre vie, in UDP il trasferimento di dati inizia senza alcun ritardo di connessione e senza dover inviare e ricevere datagrammi aggiuntivi per l'handshake. L'affidabilità del trasporto è implementata a livello applicazione tramite i meccanismi di riscontro e di ritrasmissione del sottolivello *Messaggi* del protocollo CoAP.

Connessione senza stato. TCP mantiene lo stato della connessione negli host, allocando dunque buffer di ricezione/spedizione, parametri di controllo della congestione e numeri di sequenza e riscontro. UDP non tiene traccia di questi parametri e lascia al nodo maggiori risorse disponibili per ospitare più client attivi e da la libertà di implementare una logica di trasmissione affidabile come CoAP meno onerosa in termini di risorse.

TCP

Il protocollo TCP realizza a differenza di UDP un servizio di trasporto orientato alla connessione. Prima infatti che due host possano iniziare a scambiare dati deve essere avviata una procedura di *handshake* che crea una connessione il cui stato risiede nei due terminali. Tale procedura si

[8]Sebbene molti protocolli dello strato di collegamento (Data link layer) implementino già un controllo d'errore, non c'è garanzia che un pacchetto IP, che può transitare per link con qualsiasi protocollo di collegamento, nel percorso tra sorgente e destinazione incontri solo link che forniscano il controllo di errore. É dunque utile che il protocollo di trasporto fornisca il controllo d'errore come misura di sicurezza.

compone, nell'*handshake a tre vie*, di tre operazioni: l'invio di un segmento SYN (avente cioè il campo SYN posto ad 1 nell'header di figura 2.14) dall'host che vuole stabilire la connessione, la risposta a questo con un segmento SYNACK (con i campi SYN ed ACK posti ad 1) e l'invio del riscontro del segmento SYNACK tramite un segmento ACK, inviato il quale l'host può iniziare a trasferire dati.

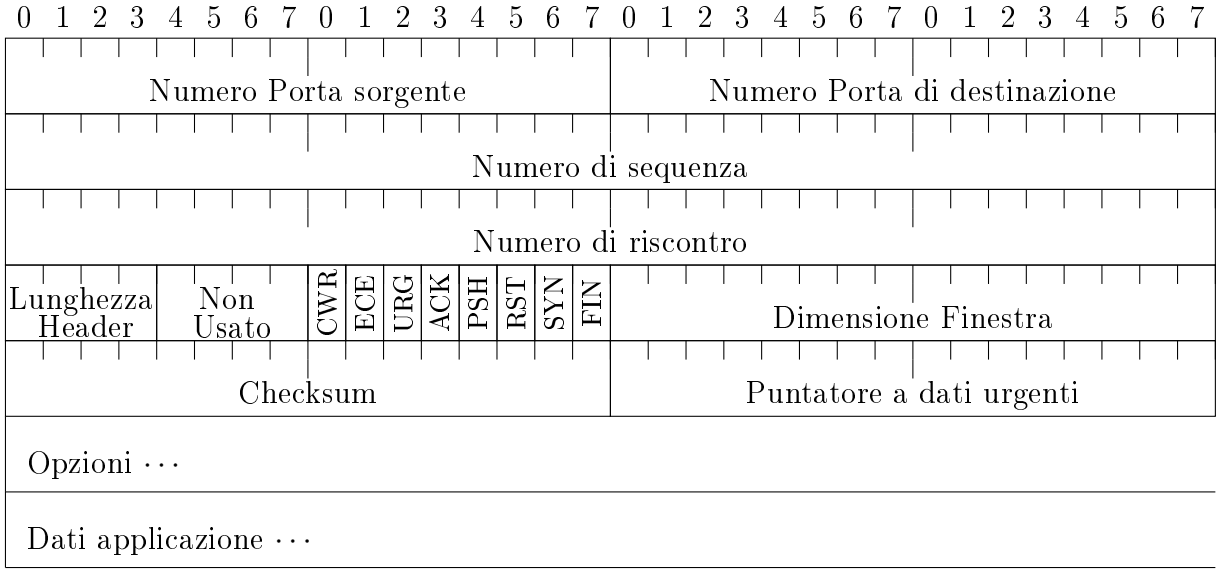


Figura 2.14: Formato del segmento TCP.

Il protocollo TCP utilizza un riscontro *cumulativo* dei byte ricevuti ovvero riscontra soltanto i byte ricevuti fino al primo mancante. In tal modo se una porzione del flusso di byte originario è mancante o è andata persa ed è stata ricevuta una sequenza di byte non adiacente all'ultimo byte richiesto, TCP invia il riscontro di ricezione dell'ultimo byte della sequenza originaria ricevuto provocando dunque la ritrasmissione da parte dell'altro host dei byte che non hanno ricevuto riscontro. Questo permette di ricevere i vari segmenti TCP come un flusso *ordinato* di byte senza preoccuparsi della gestione di eventuali segmenti persi o arrivati fuori ordine, come invece fa CoAP affidandosi su un livello di trasporto non affidabile.

Il modo con cui TCP ripristina i dati perduti è tramite l'impiego di un timeout basato sulla stima del Round Trip Time e di ritrasmissioni. La statistica del RTT che il protocollo inferisce dalle varie misurazioni dei singoli RTT di ogni trasmissione di segmenti riscontrati è caratterizzata memorizzando la sua *media mobile esponenziale ponderata* $\mathtt{m}_{\mathrm{RTT}}$ e la sua deviazione standard σ_{RTT} da cui imposta un timer T_{rtx} = di attesa di riscontro, allo scadere del quale viene effettuata una ritrasmissione, pari a

$$T_{\mathrm{rtx}} = \mathtt{m}_{\mathrm{RTT}} + 4\sigma_{\mathrm{RTT}}\,.$$

Capitolo 3

Modulo WiFi RTX4100 a basso consumo

RTX4100 è il codice prodotto di un modulo realizzato dalla società danese RTX[1] di dimensioni ridotte (18 mm × 30 mm) contenente un modulo a basso consumo Wi-Fi 802.11b/g/n a singolo stream ed un processore a basso consumo con architettura Cortex-M3. Il segmento di destinazione di tale dispositivo, illustrato in figura 3.1, è per applicazioni che trasmettano pacchetti all'interno di una rete in modo poco frequente, come ad esempio i sensori a basso consumo.

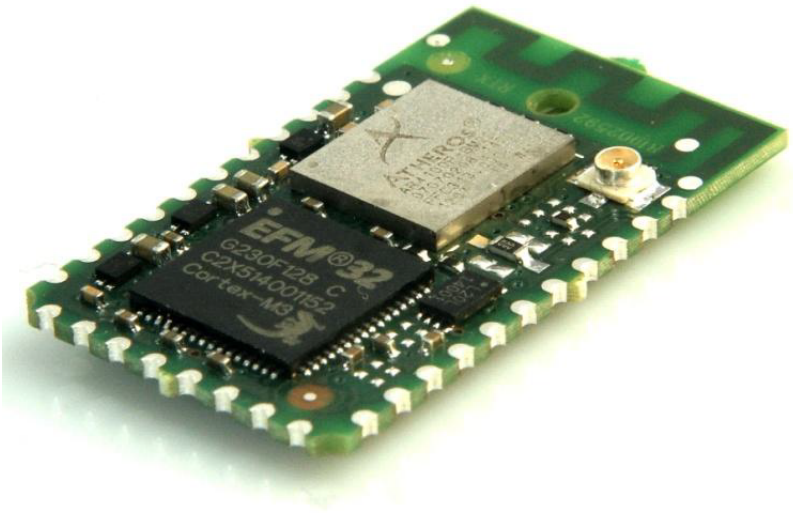


Figura 3.1: Modulo Wi-Fi RTX4100.

3.1 Caratteristiche

Il modulo RTX4100 è stato ottimizzato nei consumi per applicazioni domestiche, aziendali, smart grid, di automazione e controllo che richiedono flussi di dati piuttosto bassi ed eseguono trasmissioni o ricezioni di dati poco frequentemente. Il modulo utilizza infatti due componenti principali a basso consumo: un microcontrollore Energy Micro Gecko EFM32G230F128 ed un System in Package Wi-Fi Atheros AR4100 a singolo stream 802.11b/g/n. L'utilizzo di questa coppia di moduli a basso consumo rende RTX4100 altamente efficiente: il consumo di corrente determinato dal solo MCU (a chip WiFi spento) è infatti dell'ordine di 2.7 μA e in tale stato il processore può comunque controllare le periferiche come i sensori. I consumi del SiP AR4100 dipendono invece dalla modalità operativa e dalla attività Wi-Fi presente e sono illustrati nella sezione 3.3.

[1] http://www.rtx.dk

3.2 Architettura hardware

Un diagramma a blocchi del modulo RTX4100 è riportato in figura 3.2. I componenti e le caratteristiche principali che il modulo offre sono

- Conformità allo standard 802.11 b/g/n a singolo stream a 2.4 GHz;
- supporto alle connessioni autenticate con WEP, WPA/WPA2;
- processore Energy Micro Gecko EFM32G230F128 progettato per operazioni a bassissimo consumo energetico;
- 128 kB di memoria FLASH e 16 kB di RAM;
- modulo Atheros AR4100 Wi-Fi SiP a basso consumo avviato da una memoria FLASH seriale e comunicante tramite SPI con il MCU;
- possibilità di impiego dell'antenna integrata nel modulo Atheros oppure di una antenna esterna grazie ai connettori U.fl ed edge
- possibilità di alimentazione a batteria ed in genere da una fonte di alimentazione non regolata grazie alla presenza di regolatori di tensione LDO;
- dimensioni di 18 mm×30 mm;
- compatibilità IPv4 e IPv6;
- vasto assortimento di porte e connettori come porte ADC, DAC, comparatori analogici, porte GPIO, interfaccia SPI, I2C e porte UART.

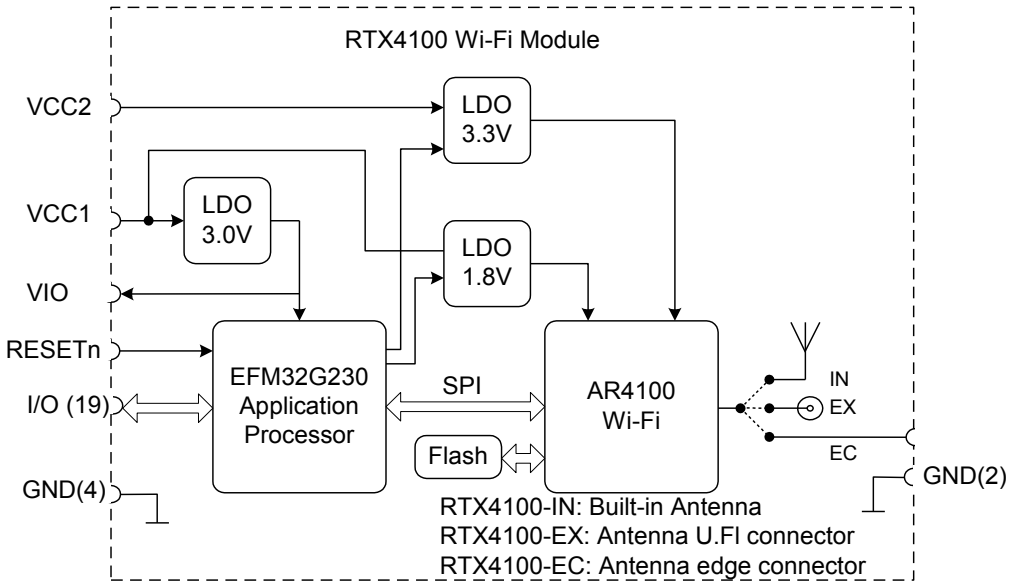


Figura 3.2: Diagramma a blocchi del modulo RTX4100.

Il modulo è venduto già saldato all'interno di una board di supporto RTX4100-WSAB (Wi-Fi Sensor Application Board) illustrata in figura 3.3. Le sue caratteristiche sono le seguenti.

Modulo RTX4100 Contiene la scheda RTX4100 descritta sopra.

Due pulsanti Sono disponibili due pulsanti comunicanti con i PIN del modulo RTX4100 che possono essere usati per applicazioni o per sviluppo.

LED bicolore É disponibile una coppia di LED (rosso e verde) controllabile dai PIN della scheda RTX4100.

Punti di saldatura esterni Vi sono terminazioni per la connessione a trasduttori resistivi connessi agli ingressi del comparatore analogico e dell'ADC. Possono essere usati per connettere ad esempio fotodiodi;

MEMS É disponibile un accelerometro a 3 assi ed una bussola per applicazioni di monitoraggio di posizione e orientamento.

Connettore di espansione/debug É disponibile un connettore che garantisce accesso a tutti i pin I/O della scheda (tranne che per il pin 29) utilizzabile per aggiungere ad esempio nuove nuovi sensori o per la connessione ad un dispositivo USB per il debug.

Alimentazione Il dispositivo può essere alimentato a batteria o tramite collegamento ad un dock aggiuntivo (RTX4100 WSAB Dock) alimentato tramite porta USB.

Resistore per misure di corrente Un resistore da 0.1 Ω (tolleranza dell'1 %) è presente in serie nel cammino verso massa dal terminale VBAT− di alimentazione a batteria per misure di corrente tramite sonda passiva.

Dimensioni contenute Le dimensioni sono di appena 25 mm×53 mm compreso il modulo RTX4100.

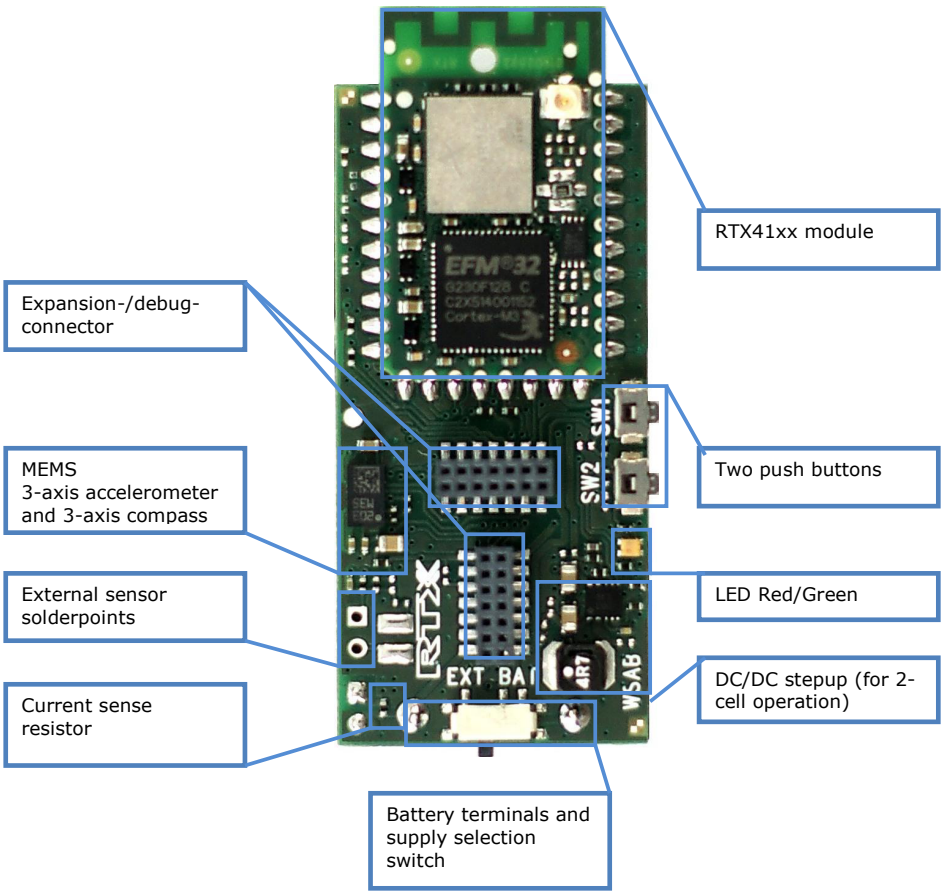


Figura 3.3: RTX4100 Wi-Fi Sensor Application Board.

3.3 Consumi energetici

La gestione dell'alimentazione (figura 3.4) del modulo RTX4100 è effettuata automaticamente dal firmware RTX OS. Il processore EFM32G230F128 viene automaticamente fatto passare dallo stato RUN MODE allo stato DEEP SLEEP MODE quando il sistema operativo rileva che non vi sono operazioni che richiedono l'attività della CPU. All'arrivo di una richiesta di interrupt, come ad esempio l'arrivo di un timeout di un timer di sistema oppure la ricezione di un evento dal chip Wi-Fi, il processore verrà riportato allo stato RUN MODE. La CPU può inoltre spegnere il modulo RF AR4100 quando non è richiesta attività Wi-Fi risparmiando così energia.

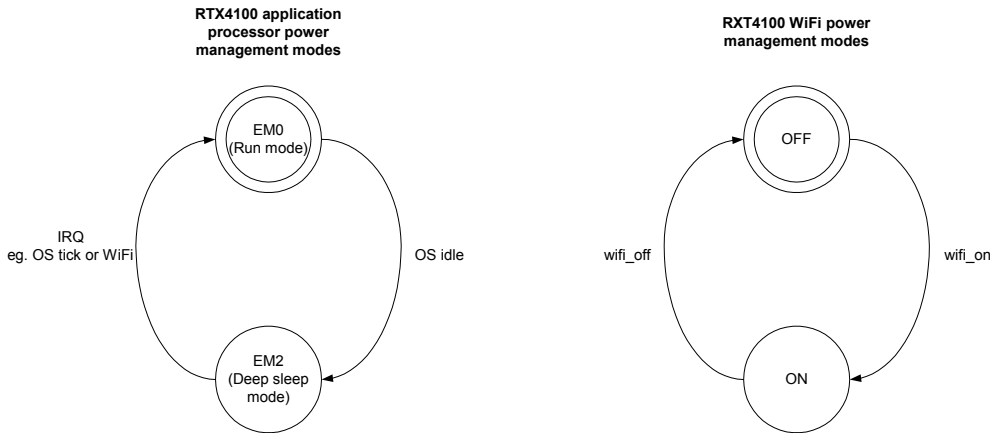


Figura 3.4: Gestione dell'alimentazione del modulo Wi-Fi RTX4100.

3.4 Architettura software

Il modulo RTX4100 come detto contiene due componenti principali: il Micro Controller Unit (MCU) EFM32G230F128 ed il modulo WiFi AR4100.

Il firmware completo del modulo può essere concettualmente diviso in due parti come illustrato in figura 3.5:

- la applicazione utente chiamata *Co-Located Application* (CoLA);
- il firmware della piattaforma Amelie.

La applicazione può essere sviluppata accedendo alle varie funzionalità offerte dal firmware della piattaforma Amelie. Il metodo con cui si può accedere a tali funzioni è tramite

- chiamate ai metodi disponibili nell'interfaccia del framework CoLA;
- Mail di richiesta/risposta inviate a o ricevute dalle primitive API offerte dal firmware Amelie.

Una volta che l'applicazione è stata scritta, è sufficiente trasferirla all'interno della memoria FLASH del modulo per poterla eseguire senza dover modificare il resto del firmware Amelie. La compilazione ed il linking della applicazione è infatti eseguito come un programma separato che a tempo di esecuzione è caricato ed eseguito come un task dal Sistema Operativo RTX OS.

Il sistema operativo proprietario RTX OS (ROS) è un sistema cooperativo basato su un sistema di scambio di messaggi (Mail) che permette ai singoli task di comunicare tra loro. Il framework CoLA è implementato come un singolo task che riceve eventi solo quando vi è una mail a lui destinata. Le applicazioni CoLA sviluppate devono essere realizzate come applicazioni completamente guidate da eventi.

Ciascuna Mail mandata all'interno del ROS contiene un identificativo di primitiva e un numero variabile di parametri. L'identificatore della primitiva può indicare un evento di sistema, un timeout oppure un evento creato dallo sviluppatore. Le Mail sono inoltrate a tutti i task attivi ed il task CoLA può mandare anche Mail a se stesso. Questo è utile quando un protothread [2] necessita di comunicare con un altro protothread in attesa di un particolare evento.

Interfaccia CoLA L'interfaccia CoLA rende disponibili all'applicazione un sottoinsieme delle funzionalità offerte dall'ambiente proprietario RTX OS (RTX Core Interface) tramite un blocco di controllo e di scambio dati che si trova in una posizione fissa all'interno della memoria FLASH. Le funzioni offerte dalla interfaccia CoLA permettono di accedere a vari moduli del firmware della piattaforma che implementano la gestione delle mail (tra applicazione CoLA e firmware), dei timer, della allocazione dinamica della memoria, della memorizzazione nella partizione non volatile della memoria FLASH interna al MCU, dell'alimentazione e delle comunicazioni seriali per i log e gli output a schermo.

Primitive API Le primitive offerte dal firmware della piattaforma Amelie sono di 4 tipi: Richiesta (REQ), Indicazione (IND), Risposta (RES) e Conferma (CFM). Una mail di tipo CFM (dallo strato API a CoLA) può giungere solo a conferma di una richiesta inviata da una primitiva di tipo REQ (da CoLA allo strato API). Una mail di tipo RES (da CoLA allo strato API) può giungere solo dopo una primitiva di tipo IND (dallo strato API a CoLA).

Le API offerte permettono la gestione delle funzionalità del firmware (per aggiornamenti), del chip Atheros Wi-Fi, della creazione e gestione socket, dello strato client/server HTTP, dello strato di trasporto TCP/UDP, della configurazione IPv4/IPv6 e del client DNS.

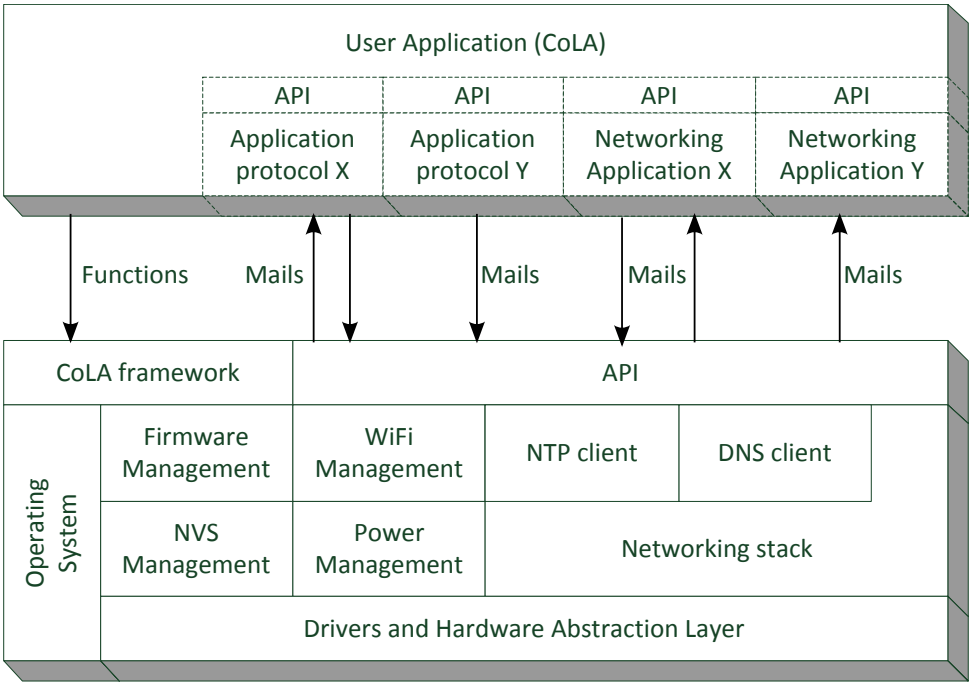


Figura 3.5: Architettura software del modulo RTX4100.

Capitolo 4

Implementazione del protocollo CoAP su RTX4100

Viene presentata in questo capitolo una descrizione dettagliata della struttura implementativa utilizzata per la realizzazione di un Client/Server CoAP eseguibile sulla scheda RTX4100. Il codice si compone di due parti fondamentali: il Parser (sezione 4.1) e il Session Formatter con macchina a stati (sezione 4.2). A queste due si aggiunge uno strato di gestione delle risorse, che non viene analizzato nel dettaglio essendo trasparente al protocollo CoAP. Esso è responsabile delle operazioni di memorizzazione, gestione ed ottenimento della rappresentazione delle risorse. Le risorse sono memorizzate secondo una struttura ad albero binario in cui ciascun nodo è identificato da una stringa costituente una porzione del campo *path* o *query* dell'URI di una risorsa ovvero ciascuna parte separata da un carattere `/`, `?` o `&`. Ciascun nodo contiene inoltre un puntatore ad un sottoalbero binario i cui nodi si riferiscono a risorse i cui URI *path* o *query* condividono la porzione iniziale con quella della radice.

4.1 Parser

Un messaggio CoAP è memorizzato nei vari nodi come una struttura dati contenente

- l'header CoAP organizzato come una struttura contenente Versione, Tipo, Lunghezza del Token, Codice messaggio e Message ID;
- una variabile `token` di lunghezza $0 \div 8$ byte;
- una variabile `uri` di lunghezza modificabile contenente la stringa URI completa della richiesta;
- una struttura contenente delle eventuali opzioni del messaggio CoAP tra quelle indicate in tabella 2.5;
- una struttura che contiene lunghezza e puntatore al payload del messaggio CoAP.

La traduzione di questa struttura in una sequenza di byte che rispetti gli standard indicati nel capitolo 2 e viceversa è realizzata dal Parser che fornisce due funzioni: `send` (sezione 4.1.1) e `receive` (sezione 4.1.2).

4.1.1 Send

Il compito della funzione `send` del parser è quella di tradurre il messaggio CoAP, codificato in una struttura a metadati a livello applicazione, in una sequenza di byte ordinata secondo il *network*

byte order (ovvero *big-endian*) all'interno di un buffer di destinazione che costituirà il payload del datagramma UDP da inviare.

La prima operazione del parser è quella di aggiungere all'inizio del buffer di destinazione (nei primi 4 byte) l'header del messaggio CoAP, includendo Versione, Tipo, Token Length, Codice e Message ID ciascuno nella propria maschera di bit (sezione 2.2) all'interno di ciascun byte dell'header.

Successivamente se il token del messaggio CoAP da inviare non è di lunghezza nulla viene scritto di seguito all'header il valore del token.

Il parser scorre quindi le opzioni presenti nella struttura opzioni del metadato CoAP nell'ordine determinato dal loro Numero di opzione riportato nella tabella 2.5 e le inserisce in successione nel buffer di destinazione previa scrittura dei byte relativi al Delta Opzione e Lunghezza Opzione.

Si noti che, come indicato nella tabella 2.5, i vari campi che compongono l'identificatore URI, Uri-Path e Uri-Query, hanno numeri di opzione 11 e 15 e tra questi vi sono i numeri delle opzioni Content-Format (12) e Max-Age (14). Pertanto la scomposizione dell'URI in una lista di opzioni Uri-Path e Uri-Query, che comincia con l'inserimento di opzioni Uri-Path (separate nell'URI dal carattere `/`), va interrotta non appena venga individuato un carattere `?` o `&` per inserire le eventuali opzioni Content-Format e Max-Age prima di poter continuare con l'inserimento delle opzioni Uri-Query.

Infine se è presente un payload di lunghezza non nulla questo viene scritto nel buffer previo inserimento del Payload Marker 0xFF.

4.1.2 Receive

La funzione `receive` del parser svolge il compito inverso della `send` traducendo il buffer di ricezione in una struttura metadati.

Inizialmente vengono copiati i valori dei vari campi Versione, Tipo, Token Length, Codice e Message ID presenti nei primi 4 byte nella struttura header del metadato CoAP. Se il campo TKL è non nullo vengono copiati i TKL byte successivi nella variabile token del metadato ed impostata la sua lunghezza a TKL.

I passi successivi consistono nella decodifica degli eventuali campi opzioni e payload, quest'ultimo contraddistinto dal payload marker. Si noti che il payload marker 0xFF può essere presente anche all'interno dei sottocampi token, Lunghezza Opzione Estesa $OL^{(ex)}$ o Valore Opzione, pertanto una semplice ricerca byte a byte del marker 0xFF risulta un metodo non affidabile per l'individuazione dell'inizio del campo payload.

Dopo i primi 4 byte dell'header, ed i TKL byte del campo token, il campo in cui *non* può essere presente il payload marker è il byte iniziale di ciascun campo opzione, i cui primi 4 bit formano il campo $OD^{(0)}$ ed i restanti 4 formano il campo $OL^{(0)}$. Ciascuno di questi due campi, come detto nella sezione 2.2.1, può valere al massimo 14 eliminando così la possibilità che il byte complessivo sia pari ad 0xFF. Pertanto, al termine della copia dell'eventuale token, il parser mantiene un puntatore `ptr` al byte successivo all'ultimo byte del token (o dell'header nel caso di TKL nulla):

1. se il valore del byte puntato è 0xFF allora il parser interpreta i rimanenti byte del datagramma come payload;

2. altrimenti interpreta il byte puntato come byte iniziale di un campo opzione da cui può ricavare Delta e Lunghezza dell'opzione corrente (eventualmente estese nei byte successivi); dopo aver decodificato e copiato nel metadato l'opzione corrente, il puntatore `ptr` viene incrementato dello stesso numero di byte del campo opzione (compresi i byte dei sottocampi $OD^{(0)}$, $OL^{(0)}$, $OD^{(ex)}$, $OL^{(ex)}$) appena decodificata dal buffer e viene rieseguita la procedura dei punti 1 e 2 fino a quando non si arriva all'ultimo byte del datagramma.

4.2 Session Formatter e Macchina a stati

Come già detto, a differenza di HTTP che è un protocollo *stateless*, CoAP implementa funzionalità come il trasporto affidabile senza appoggiarsi alla logica del protocollo TCP ma impiegando una propria logica a stati.

Ciascuna sessione CoAP contiene, oltre che ad una propria macchina a stati, i seguenti parametri:

- una variabile `session_mid` che memorizza il Message ID della sessione corrente; esso può venire aggiornato nel caso di una risposta di tipo Separate per la quale è previsto l'invio di messaggio CON con un nuovo Message ID;
- una variabile `session_token` che memorizza il Token della richiesta/risposta gestita nella sessione corrente, usata per associare, tramite il filtro di ricezione (si veda il paragrafo successivo), ogni messaggio di risposta ricevuto alla corretta sessione che è in attesa e per inviare, dal lato server, una risposta con lo stesso token presente nella richiesta originaria che ha avviato la sessione;
- l'indirizzo IP e la porta dell'host remoto con cui si scambiano i messaggi, nonché il numero di socket associato localmente a tale host;
- un timer utilizzato per la temporizzazione delle permanenze nei vari stati.

Le macchine a stati per le sessioni Client e Server sono trattate separatamente nelle prossime due sezioni. Gli eventi che causano una transizione della macchina a stati sono di 3 tipi:

- `SEND`: una richiesta di invio di un messaggio CoAP che può arrivare dalla applicazione quando inizia una nuova sessione Client oppure dallo strato di gestione delle risorse quando ottiene la rappresentazione della risorsa richiesta dall'host remoto e la recapita alla sessione attiva che è in attesa di trasmetterla;
- `RECEIVE`: la ricezione di un pacchetto UDP destinato ad un socket di una sessione attiva;
- `TIMER_FIRED`: l'arrivo dell'indicazione di timeout di un timer precedentemente avviato dalla sessione stessa.

All'arrivo di uno di questi eventi verrà chiamata la corrispondente funzione da eseguire in base allo stato in cui si trova la sessione.

Filtro di ricezione Alla ricezione di un datagramma UDP, il parser traduce la sequenza di byte del buffer di ricezione in un pacchetto CoAP che viene passato a tutte le sessioni attive. Ciascuna sessione attiva esegue una analisi del pacchetto tramite la funzione `accept` il cui compito è di verificare se il pacchetto è destinato o meno alla corrispondente sessione attiva. Le operazioni che `accept` svolge sono le seguenti:

- si verifica che indirizzo IP di destinazione e porta UDP di destinazione del datagramma ricevuto coincidano con quelli della sessione attiva;
- se la precedente condizione è verificata si guarda il tipo e il codice del messaggio:
  - se il codice è 0 e il tipo è ACK o RST (riscontro vuoto) allora il pacchetto viene accettato se ha Message ID coincidente con quello della sessione attiva;
  - se il tipo è ACK ma il codice non è 0 (riscontro non vuoto con risposta piggy-backed) allora il pacchetto viene accettato se sia il suo Message ID che il suo token coincidono con quelli della sessione attiva;

- se il tipo è diverso da ACK (CON, NON o RST) e il codice è non nullo, allora il pacchetto viene accettato se il suo token coincide con quello della sessione attiva;
- se nessuna delle precedenti condizioni è verificata, il pacchetto viene scartato.

Generazione Message ID La modalità con cui viene generato il Message ID da utilizzare per i messaggi delle varie sessioni è basata sull'indirizzo MAC del nodo che genera la richiesta e il tempo in secondi.

All'avvio del nodo viene generato un intero `last_mid` a 16 bit tramite un generatore di numeri pseudo-casuale che utilizza come *seed* per la generazione un intero ottenuto con una funzione hash a partire dall'indirizzo MAC del nodo che genera la richiesta e il tempo in secondi in cui viene avviata la macchina. Il tempo è ottenuto tramite protocollo NTP inviando una richiesta ad un server NTP.

Una volta che la variabile statica `last_mid` è stata generata, ogni sessione può richiedere un nuovo Message ID semplicemente incrementando di 1 unità tale variabile.

4.2.1 Client CoAP

Un nodo CoAP che voglia inviare una richiesta CoAP richiede la creazione di una sessione Client inizializzata con il proprio socket e la macchina a stati Client riportata in figura 4.1.

La sessione Client viene inizializzata con token, indirizzo remoto e socket del pacchetto che la applicazione consegna al Session Formatter al momento della richiesta di invio.

Stato IDLE

Lo stato IDLE è lo stato in cui si trova inizialmente la macchina a stati di una sessione client.

`SEND` All'arrivo di una richiesta SEND dall'applicazione, dopo la creazione di una nuova sessione e la sua inizializzazione come indicato sopra, viene salvato indirizzo IP e porta di destinazione nelle variabili statiche della sessione aperta e viene aperto un socket locale, il cui numero viene salvato nelle variabile corrispondente della sessione avviata. Viene quindi invocata la funzione `send` dello stato IDLE che esegue le seguenti operazioni a seconda del Tipo della richiesta CoAP (CON,NON).

- Per una richiesta CON viene assegnato alla sessione ed al pacchetto CoAP da inviare un nuovo Message ID (generato secondo la regola indicata sopra), viene inviato il pacchetto all'indirizzo IP e porta memorizzati nella sessione (previa scrittura del pacchetto nel buffer tramite la funzione `send` del parser), viene salvato il pacchetto nello stato WAIT ACK per eventuali ritrasmissioni in caso di mancato riscontro e viene avviato il timer di ritrasmissione T_0^{rtx} come indicato nella 2.1 oltre ad inizializzare a zero il numero di ritrasmissioni: $N_{\text{rtx}} = 0$. La macchina a stati transita quindi nello stato WAIT ACK.
- Per una richiesta NON viene assegnato alla sessione ed al pacchetto CoAP da inviare un nuovo Message ID e viene inviato il pacchetto all'indirizzo IP e porta memorizzati nella sessione. A differenza del caso NON in questo caso il pacchetto non viene salvato per ritrasmissioni, trattandosi di una richiesta Non-Confermabile, ma viene semplicemente avviato un timer $T_{\text{srv resp}}$ di attesa della risposta NON da parte del server. La macchina a stati viene fatta passare allo stato WAIT SERVER RESPONSE NON.

Stato WAIT SERVER RESPONSE NON

Lo stato WAIT SERVER RESPONSE NON è lo stato nel quale il client si trova in attesa di una risposta NON ad una richiesta precedentemente inviata di tipo NON.

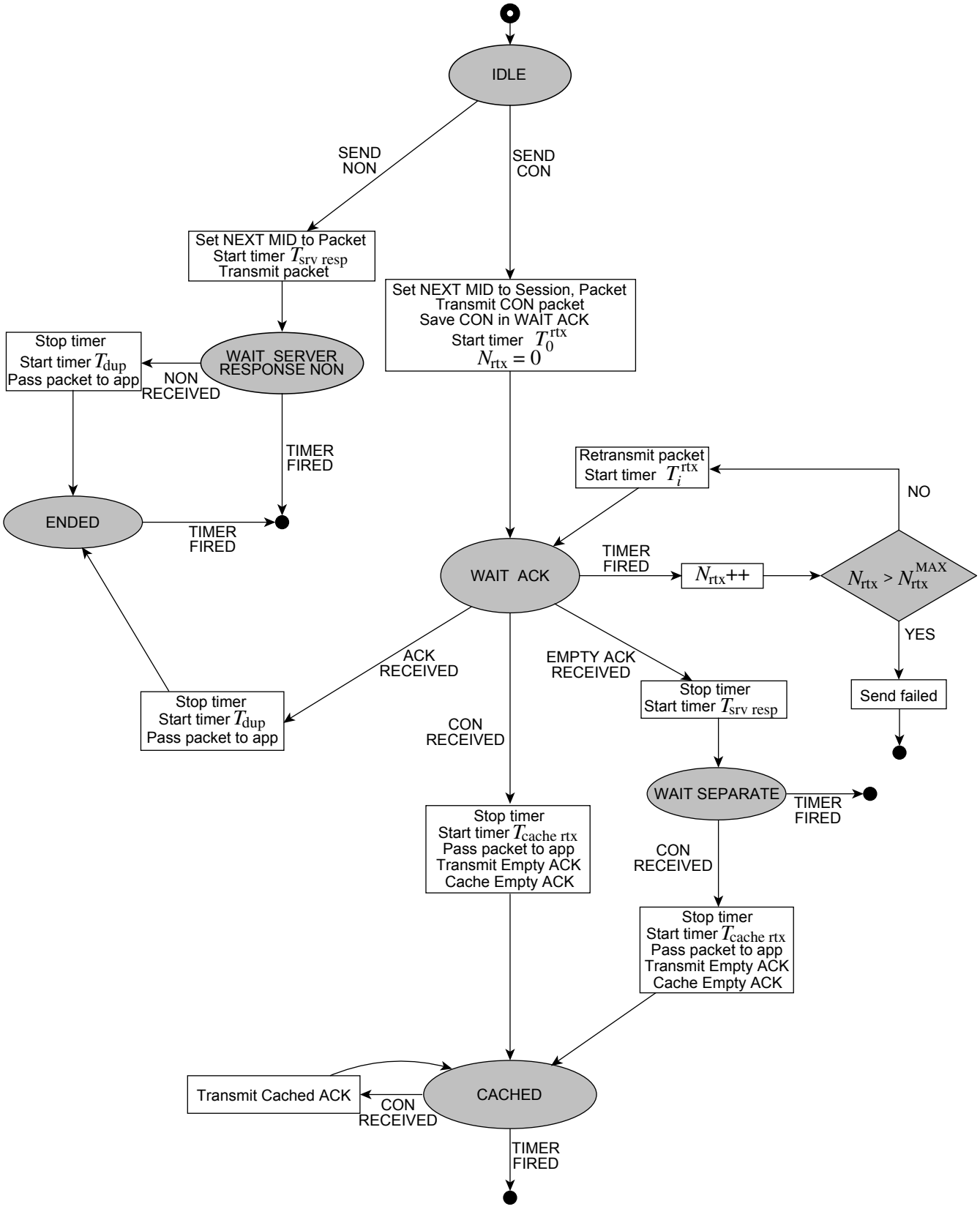


Figura 4.1: Macchina a stati di una sessione Client CoAP.

`RECEIVE` Alla ricezione di un messaggio di risposta di tipo NON (accettato dal filtro di ricezione) questo viene passato alla applicazione che ha prodotto la richiesta originaria e viene avviato un timer T_{dup} di attesa di eventuali arrivi di messaggi duplicati o non in ordine. La sessione passa allo stato ENDED.

`TIMER_FIRED` Alla ricezione del timeout del timer $T_{\text{srv resp}}$ avviato dallo stato precedente la sessione viene semplicemente terminata.

Stato WAIT ACK

Lo stato WAIT ACK è lo stato principale di gestione delle ritrasmissioni una richiesta Confermabile. In tale stato la sessione giunge per la pirma volta dopo aver inviato messaggio CON di richiesta ed aver avviato il timer di prima ritrasmissione T_0^{rtx}.

`TIMER_FIRED` Allo scadere del timer T_i^{rtx} si incrementa di 1 unità il contatore N_{rtx}. Se il numero di ritrasmissioni effettuate N_{rtx} è superiore al limite $N_{\text{rtx}}^{\text{MAX}}$ la sessione viene terminata e si segnala alla applicazione il fallimento della operazione di ottenimento della risposta, altrimenti si procede alla ritrasmissione della richiesta CON che non ha ricevuto riscontro, si avvia un nuovo timer T_i^{rtx}, con $i = N_{\text{rtx}}$, e si rimane nello stato WAIT ACK senza effettuare alcuna transizione di stato.

`RECEIVE` Alla ricezione di un messaggio CoAP vengono eseguite le seguenti operazioni:

- se il messaggio è un ACK con risposta *piggy-backed*, si arresta il timer corrente, viene passata la risposta alla applicazione che ha generato la richiesta e viene avviato un timer T_{dup} per poi passare allo stato ENDED;
- se il messaggio è un ACK vuoto il client si aspetta una prossima ricezione di una risposta *separate*, pertanto viene fermato il timer corrente ed avviato un timer $T_{\text{srv resp}}$ di attesa di tale risposta mentre la macchina a stati passa allo stato WAIT SEPARATE;
- se il messaggio è una risposta CON, ovvero di tipo *separate*, arrivata prima della ricezione dell'ACK vuoto, il client interpreta tale situzione come la mancata ricezione dell'ACK vuoto che è andato perso o potrebbe giungere dopo; procede quindi al recapito del messaggio ricevuto alla applicazione, ferma il timer corrente, trasmette al client un ACK vuoto di riscontro della risposta separate e memorizza tale pacchetto nello stato CACHED al quale la macchina a stati passa dopo aver avviato il timer $T_{\text{cache rtx}}$.

Stato WAIT SEPARATE

Lo stato WAIT SEPARATE è lo stato nel quale la sessione si trova quando è stato ricevuto un ACK vuoto e si sta attendendo la risposta *separate*.

`RECEIVE` Alla ricezione della risposta CON *separate* questa viene recapitata alla applicazione che la sta attendendo, si termina il timer $T_{\text{srv resp}}$ precedentemente avviato e si avvia un nuovo timer $T_{\text{cache rtx}}$ dopo aver trasmesso un ACK vuoto di riscontro della risposta separate e memorizzato tale pacchetto nello stato CACHED. Si passa quindi allo stato CACHED.

`TIMER_FIRED` All'arrivo del timeout $T_{\text{srv resp}}$ si conclude l'attesa della risposta *separate* e viene terminata la sessione.

Stato CACHED

Lo stato CACHED è lo stato nel quale la macchina a stati del client si viene a trovare dopo che è stata ricevuta una risposta *separate*. La sua funzione è quella di ritrasmettere i riscontri ACK della risposta *separate* qualora giungessero altri messaggi CON duplicati o ritrasmessi dal server che non abbia ricevuto l'ACK.

`RECEIVE` Quando viene ricevuto un messaggio CON lo stato provvederà semplicemente a ritrasmettere il messaggio di ACK già inviato senza avviare successive transizioni della macchina a stati per poter ritrasmettere nuovi riscontri a messaggi CON che giungano entro $T_{\text{cache rtx}}$.

`TIMER_FIRED` Allo scadere del timer $T_{\text{cache rtx}}$ la sessione viene terminata.

Stato ENDED

Lo stato ENDED è lo stato nel quale la sessione client si trova dopo aver ricevuto il messaggio di risposta NON ad una richiesta NON o il riscontro ACK con risposta *piggy-backed* di una richiesta CON.

`TIMER_FIRED` Alla ricezione del timeout del timer T_{dup} avviato dallo stato precedente la sessione viene semplicemente terminata.

Si noti che sebbene lo stato ENDED non sia previsto per nessuno scopo tranne che l'attesa del timeout T_{dup}, esso ha la funzione di mantenere la sessione attiva in modo che il socket associato rimanga aperto ed eventuali messaggi ricevuti duplicati o arrivati in ordine modificato destinati a tale socket vengano semplicemente accettati e ignorati (senza eseguire alcuna azione), evitando così che tali pacchetti generino l'apertura dal lato server di una nuova sessione.

4.2.2 Server CoAP

Al momento della ricezione di un datagramma UDP, il parser lo traduce in un pacchetto CoAP che viene passato a tutte le sessioni attive. Se nessuna sessione attiva accetta il pacchetto (ovvero indirizzo IP e porta UDP sorgenti non coincidono con nessuna delle sessioni aperte) viene avviata una nuova sessione Server che gestirà la nuova richiesta ricevuta avviando la funzione `receive`.

Un nodo CoAP che voglia inviare una richiesta CoAP richiede la creazione di una sessione Client inizializzata con il proprio socket e la macchina a stati Client riportata in figura 4.1.

La sessione Server viene inizializzata, oltre che con la propria macchina a stati Server riportata in figura 4.2, con token, indirizzo IP e porta remoti del pacchetto ricevuto.

Stato IDLE

Lo stato IDLE è lo stato di inizializzazione della macchina a stati di una sessione server.

`RECEIVE` All'arrivo dell'indicazione di ricezione della richiesta CoAP, dopo la creazione della nuova sessione e la sua inizializzazione come indicato sopra, la funzione `receive` dello stato IDLE esegue le seguenti operazioni a seconda del Tipo della richiesta CoAP ricevuta (CON,NON):

- per una richiesta CON viene assegnato alla sessione il Message ID del pacchetto ricevuto, viene inoltrata la richiesta allo strato di gestione delle risorse e viene avviato il timer $T_{\text{app resp pb}}$ di attesa di ottenimento della risorsa prima dell'invio dell'ACK vuoto;
- per una richiesta NON viene inoltrata la richiesta allo strato di gestione delle risorse e viene avviato un timer di attesa $T_{\text{srv resp}}$ pari allo stesso tempo di attesa da parte del client della risposta NON allo scadere del quale il client termina la sessione.

Stato WAIT APP RESPONSE PIGGY-BACKED

Lo stato WAIT APP RESPONSE PIGGY-BACKED è lo stato in cui si trova la sessione server dopo aver inoltrato la richiesta CON ricevuta allo strato di gestione delle risorse.

`SEND` Se la rappresentazione della risorsa richiesta arriva prima della scadenza del timer $T_{\text{app resp pb}}$ si termina questo timer e si procede all'invio della risposta *piggy-backed* allegandola nell'ACK. Il Message ID che tale messaggio di risposta trasporta viene copiato da quello memorizzato nella sessione (che a sua volta è stato copiato dal messaggio di richiesta originario). La risposta viene anche memorizzata nello stato CACHED per ritrasmetterla in caso di arrivi di messaggi CON duplicati o ritrasmessi. Viene avviato infine il timer $T_{\text{cache rtx}}$, allo scadere del quale la sessione verrà terminata, e si passa allo stato CACHED.

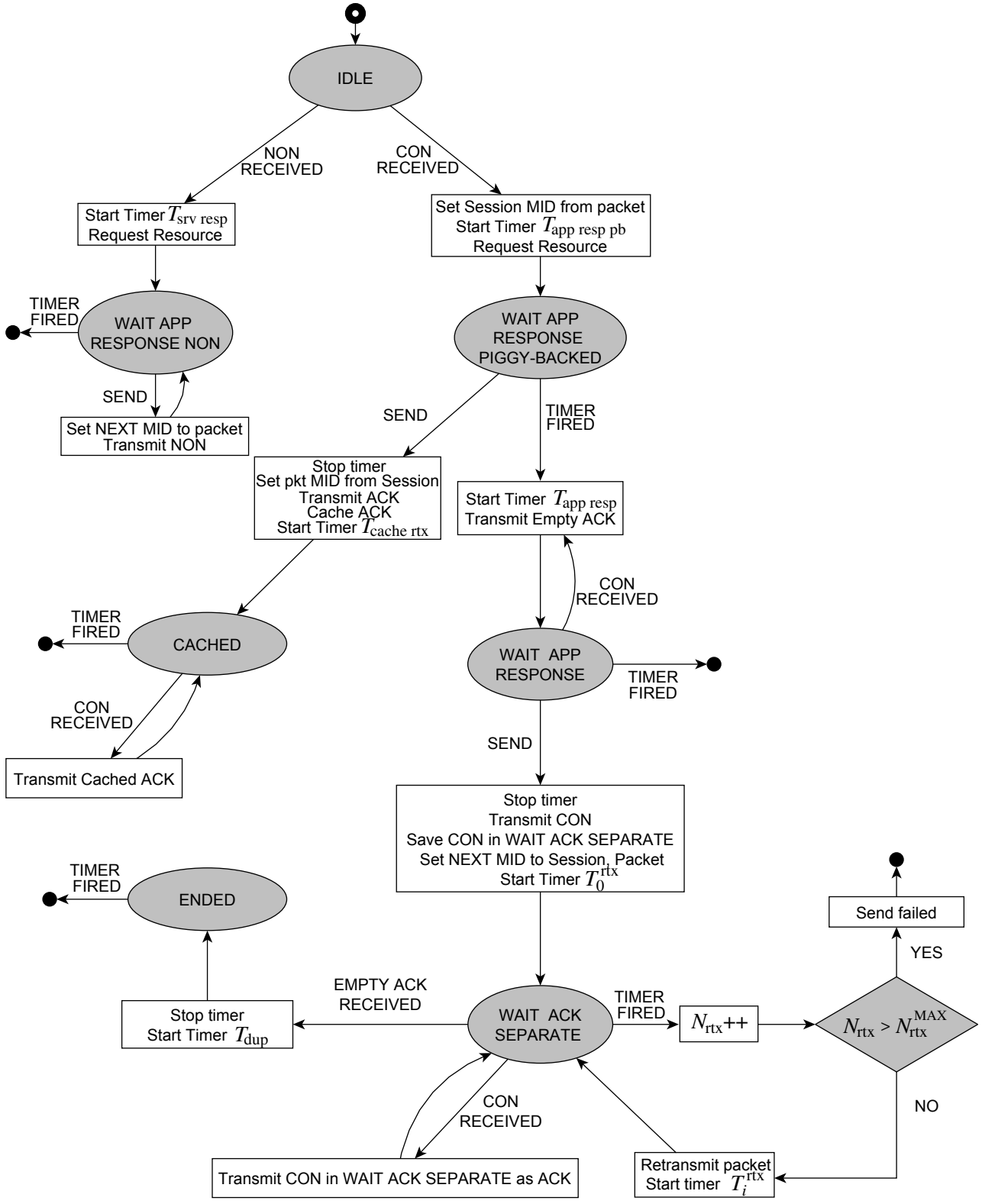


Figura 4.2: Macchina a stati di una sessione Server CoAP.

`TIMER_FIRED` Se la risorsa richiesta non giunge dallo strato di gestione delle risorse entro lo scadere del timer $T_{\text{app resp pb}}$ viene trasmesso un ACK vuoto per notificare al client la ricezione della richiesta CON ed il prossimo invio di una risposta *separate*. Si avvia quindi un timer $T_{\text{app resp}}$ di attesa della risorsa più lungo di $T_{\text{app resp pb}}$ e si passa allo stato WAIT APP RESPONSE.

É stato impostato un timer di attesa della riposta piggy-backed molto breve, in modo che nei casi in cui la risorsa non risulti disponibile entro questo breve termine il server proceda subito all'invio del riscontro ACK vuoto al client cosicché questo possa sperabilmente riceverlo

prima dell'arrivo del primo timeout di ritrasmissione T_0^{rtx} evitando così il reinvio del messaggio di richiesta.

Con i valori standard indicati nelle tabelle 2.1 e 2.3 si ha $T_{\text{RTT}}^{\text{MAX}} > T_0^{\text{rtx}}$ e non è pertanto possibile impostare un timer $T_{\text{app resp pb}}$ che garantisca sempre l'arrivo dell'ACK vuoto al client prima dell'arrivo del timeout T_0^{rtx}.

Stato WAIT APP RESPONSE

Lo stato WAIT APP RESPONSE è lo stato di attesa dell'arrivo della rappresentazione della risorsa richiesta dallo strato di gestione delle risorse dopo che è trascorso il tempo massimo $T_{\text{app resp pb}}$ di attesa per l'invio della risposta *piggy-backed*.

`SEND` Se la rappresentazione della risorsa richiesta arriva prima della scadenza del timer $T_{\text{app resp}}$ si termina questo timer e si procede all'invio della risposta *separate* in un nuovo messaggio CON. Viene generato un nuovo Message ID che verrà assegnato al Message ID di tale messaggio e a quello della sessione server corrente. Oltre ad essere trasmesso il messaggio viene anche salvato nello stato WAIT ACK SEPARATE per essere ritrasmesso nel caso di mancato riscontro. Viene avviato il timer di prima ritrasmissione T_0^{rtx} e si passa allo stato WAIT ACK SEPARATE.

`TIMER_FIRED` Se il timeout del timer $T_{\text{app resp}}$ giunge prima che lo strato di gestione delle risorse riesca ad ottenere la rappresentazione della risorsa richiesta la sessione viene terminata.

Stato WAIT ACK SEPARATE

Lo stato WAIT ACK SEPARATE è lo stato principale di gestione delle ritrasmissioni della risposta CON *separate*. In tale stato la sessione giunge per la pirma volta dopo aver inviato tale risposta ed aver avviato il timer di prima ritrasmissione T_0^{rtx}.

`TIMER_FIRED` Allo scadere del timer T_i^{rtx} si incrementa di 1 unità il contatore N_{rtx}. Se il numero di ritrasmissioni effettuate N_{rtx} è superiore al limite $N_{\text{rtx}}^{\text{MAX}}$ la sessione viene terminata e si segnala alla applicazione il fallimento della operazione di invio della risposta *separate*, altrimenti si procede alla ritrasmissione della risposta CON che non ha ricevuto riscontro, si avvia un nuovo timer T_i^{rtx}, con $i = N_{\text{rtx}}$, e si rimane nello stato WAIT ACK SEPARATE senza effettuare alcuna transizione di stato.

`RECEIVE` Alla ricezione di un messaggio CoAP vengono eseguite le seguenti operazioni:

- se il messaggio è un ACK vuoto, si arresta il timer corrente, viene segnalato alla applicazione il successo della trasmissione della risposta *separate* e viene avviato un timer T_{dup} per poi passare allo stato ENDED;
- se il messaggio è di tipo CON il server interpreta tale situazione come la mancata ricezione dell'ACK vuoto e della risposta *separate* da parte del client; procede quindi alla ritrasmissione della risposta memorizzata nello stato SEPARATE come ACK, ovvero non più come risposta *separate* ma come *piggy-backed* in quanto non è stato ricevuto l'ACK vuoto dal client e si rimane nello stato WAIT ACK SEPARATE.

Stato WAIT APP RESPONSE NON

Lo stato WAIT APP RESPONSE NON è lo stato in cui si trova la sessione server dopo aver inoltrato la richiesta NON ricevuta allo strato di gestione delle risorse.

`SEND` Se la rappresentazione della risorsa richiesta arriva prima della scadenza del timer $T_{\text{srv resp}}$ si termina questo timer e si procede all'invio della risposta in un messaggio NON. Tale messaggio di risposta verrà inviato con un nuovo Message ID generato dal server. Si rimane quindi nello stato WAIT APP RESPONSE NON in attesa della fine del timer $T_{\text{srv resp}}$, allo scadere del quale la sessione verrà terminata, e si passa allo stato CACHED. Il motivo per cui non viene terminata immediatamente la sessione è analogo a quello illustrato per il caso dello stato ENDED, ovvero si vuole mantenere aperta la sessione per ricevere eventuali richieste NON che giungano duplicate o non in ordine ed ignorarle evitando che queste avviino una nuova sessione server.

`TIMER_FIRED` Nel caso in cui la risorsa non sia disponibile entro il tempo di attesa $T_{\text{srv resp}}$ la sessione viene semplicemente terminata.

Stato ENDED

Lo stato ENDED dal lato server ha una funzione analoga a quella dello stesso stato nella macchina a stati del lato client. Allo stato ENDED la macchina a stati giunge non appena è stato ricevuto un ACK vuoto per una risposta *separate*. Si è pertanto sicuri che il client che ha inviato la richiesta ha già ricevuto la risposta *separate* in un messaggio CON avendone già ricevuto il riscontro. Tuttavia poiché l'ordine di arrivo dei datagrammi UDP non è garantito, è possibile ricevere un pacchetto ritrasmesso (o duplicato) della richiesta CON originaria anche dopo la ricezione del riscontro ACK vuoto. Grazie allo stato ENDED la sessione non viene chiusa ma rimane attiva per un ulteriore tempo T_{dup} di modo che l'eventuale arrivo dei citati pacchetti CON ritrasmessi o duplicati non provochi l'apertura di una nuova sessione server ma semplicemente il recapito di tali pacchetti alla sessione ancora aperta che li ignorerà.

`TIMER_FIRED` L'arrivo del timeout del timer T_{dup} provoca semplicemente la terminazione della sessione corrente.

Stato CACHED

Lo stato CACHED è lo stato nel quale la macchina a stati del server si viene a trovare dopo che è stata inviata una risposta *piggy-backed*. La sua funzione è quella di ritrasmettere la risposta *piggy-backed* nell'ACK qualora giungessero altri messaggi CON duplicati o ritrasmessi dal server che non abbia ricevuto l'ACK.

`RECEIVE` Quando viene ricevuto un messaggio CON si provvederà semplicemente a ritrasmettere la risposta ACK già inviata rimanendo quindi nello stato RECEIVE per poter ritrasmettere nuovi riscontri a messaggi CON che giungano entro $T_{\text{cache rtx}}$.

`TIMER_FIRED` Allo scadere del timer $T_{\text{cache rtx}}$ la sessione viene terminata.

Capitolo 5

Risultati sperimentali

5.1 IEEE 802.11 Power-Saving Mode

Come indicato nel capitolo 2 CoAP è pensato per offrire le funzionalità di un protocollo di livello applicazione come HTTP a dispositivi di rete con risorse computazionali ed energetiche limitate come sensori a basso consumo di potenza. I componenti dei nodi di rete WiFi che richiedono maggiori risorse in termini energetici sono gli amplificatori presenti nel chip RF che servono ad amplificare il segnale da trasmettere ed il segnale ricevuto. Risulta chiaro che se si vuole mantenere attivi dei nodi di rete con limitate risorse energetiche, ad esempio alimentati tramite batterie, mantenere il chip WiFi sempre attivo non risulta una soluzione efficiente per i consumi energetici e comporta un esaurimento rapido della carica residua su tali dispositivi. Lo spegnimento del ricetrasmettitore WiFi per opportuni intervalli temporali risulta dunque un metodo per poter raggiungere consistenti risparmi energetici in reti di questo tipo.

Lo standard 802.11 [6] prevede due modalità operative per le stazioni mobili all'interno di reti WLAN ad infrastruttura (Infrastructure basic service set):

- una modalità detta*Active mode* (AM) in cui la stazione può ricevere frame in qualsiasi momento e deve pertanto mantenere attivo il ricetrasmettitore radio;
- una modalità operativa detta *Power-saving mode* (PSM) atta a minimizzare i periodi di attività del ricetrasmettitore radio (periodo di attività) ed a massimizzare il tempo in cui questo è spento (periodo dormiente o in *power-save*) senza rinunciare alla connettività.

Nelle reti WLAN ad infrastruttura tutto il traffico proveniente e destinato a stazioni mobili passa attraverso un Access Point (AP) centrale. Ogni stazione mobile che desideri operare in modalità *power-save* deve comunicarlo all'Access Point che confermerà con un frame di riscontro ACK o BlockAck.

L'Access Point svolge il compito di memorizzare in un buffer tutti i frame destinati alle stazioni in PSM collegate che giungano durante gli intervalli di riposo (di non attività) delle stesse. La presenza di questi frame in attesa di recapito memorizzati nel buffer viene notificata alla stazione di destinazione in corrispondenza degli istanti in cui la stazione si riattiva periodicamente.

Una stazione che opera in modalità *power-save* può spegnere periodicamente il ricetrasmettitore radio e riaccenderlo ad intervalli regolari che vengono concordati con l'Access Point al momento della richiesta di passaggio in PSM. Tali intervalli vengono detti *Listen Interval* e sono un multiplo intero del periodo di Beacon. Il Beacon frame è un frame di gestione di livello MAC trasmesso periodicamente dall'AP formato da un MAC header, un frame body ed una sequenza di controllo d'errore FCS. Oltre ad includere un Timestamp, attualizzato al momento della trasmissione, che permette alle altre stazioni di mantenere la sincronizzazione, il Beacon Frame contiene vari campi tra cui il Beacon Interval, che indica il tempo (espresso in unità temporali TU pari a $1024\,\mu$s) che intercorre tra la trasmissione di due Beacon consecutivi ed il campo TIM (Traffic Indication Map),

che indica se nel buffer dell'Access Point sono presenti dei frame destinati a stazioni in modalità PS (dormienti).

Ciascuna stazione radio può richiedere un Listen Interval ampio per risparmiare energia ma ciò comporta maggiore latenza e richiede uno spazio di archiviazione maggiore nel buffer nell'Access Point.

Una stazione che passa dallo stato inattivo (dormiente) allo stato attivo e vuole effettuare la trasmissione di un frame deve prima richiedere al livello fisico PLCP che venga effettuata l'operazione di Carrier Sense/Clear Channel Assessment (CS/CCA) fino a quando non si verifica una delle seguenti condizioni:

- viene rilevato un frame che permetta di aggiornare il vettore di allocazione di rete NAV;
- oppure trascorre un periodo di tempo pari a ProbeDelay[1] durante il quale il canale è rimasto libero.

Se il mezzo viene rilevato libero, lo standard [6] prevede che la stazione attenda un periodo pari ad un DCF Interframe Space (DIFS). Se il canale risulta libero per tale intervallo la trasmissione del frame può avvenire, altrimenti al termine del DIFS la stazione deve attendere un ulteriore Tempo di Backoff T_{BO} generato casualmente nel seguente modo:

$$T_{\mathrm{BO}} = \rho \cdot T_{\mathrm{Slot}} \,, \tag{5.1}$$

dove $\rho \in [0, \mathrm{CW}]$ è un intero pseudocasuale generato uniformemente nell'intervallo $[0, \mathrm{CW}]$ e T_{Slot} è un intervallo dipendente dallo strato fisico. CW è detta finestra di collisione ed è un intero posto inizialmente a $\mathrm{CW}_{\mathrm{min}}$ per il primo intervallo T_{BO} e raddoppiato fino al suo valore massimo $\mathrm{CW}_{\mathrm{max}}$ ogni volta che durante un intervallo di Backoff T_{BO} si rilevi il mezzo ancora occupato. I valori di T_{Slot}, $\mathrm{CW}_{\mathrm{min}}$ e $\mathrm{CW}_{\mathrm{max}}$ previsti dallo standard IEEE 802.11 [6] per tre tipi di PHY sono riportati in tabella 5.1.

PHY	$\boldsymbol{T_{\mathrm{Slot}}}$	$\mathbf{CW_{min}}$	$\mathbf{CW_{max}}$
FHSS	$50\,\mu$s	16	1024
DSSS	$20\,\mu$s	32	1024
IR	$8\,\mu$s	64	1024

Tabella 5.1: Slot Time e valori limite della Contention Window per tre standard PHY: Frequency Hopping Spread Spectrum (FHSS), Direct Sequence Spread Spectrum (DSSS) ed Infrarosso (IR).

La modalità con cui l'Access Point informa le varie stazioni in PSM della presenza nel buffer di ricezione di frame in attesa di recapito è tramite l'inclusione di un campo detto *Traffic Indication Map* (TIM) all'interno di ogni trasmissione periodica dei Beacon. Il TIM è una mappa virtuale di 2008 byte in cui ogni bit corrisponde ad un particolare Association ID (AID[2]): quando l'Access Point riceve dei frame destinati ad una stazione precedentemente associata in PSM, imposta ad 1 il bit corrispondente all'AID di tale stazione nel TIM del prossimo Beacon.

La stazione in PSM al momento del risveglio dallo stato dormiente si mette in ascolto per ricevere il Beacon ed esamina il campo TIM: se vede che il bit corrispondente al suo AID è nullo deduce che non ci sono frame in attesa di recapito e torna allo stato dormiente, altrimenti avvia la procedura di richiesta dei frame in attesa nel buffer. Questa procedura, illustrata in figura 5.1, consiste nell'invio di un frame di controllo detto PS-Poll: ogni frame PS-Poll è utilizzato per

[1] Il ProbeDelay è l'intervallo di tempo (dell'ordine di qualche microsecondo) che una stazione in PSM che passi da stato dormiente a stato attivo deve attendere prima di trasmettere un frame nel caso non sia rilevato nessun frame che permetta di aggiornare il vettore di allocazione di rete NAV.

richiedere l'invio di *un singolo* frame in attesa nel buffer dell'AP. Alla completa ricezione di tale frame la stazione deve inviare un riscontro ACK all'AP di modo che questo possa eliminare dal buffer il frame appena recapitato.

Se più di un frame è in attesa di recapito per una stazione, l'AP imposterà ad 1 il campo More Data nel campo Frame Control. In tal modo la stazione potrà inviare ulteriori Ps-Poll per ricevere i frame successivi fino a quando il campo More Data non risulta 0, come illustrato in figura 5.1.

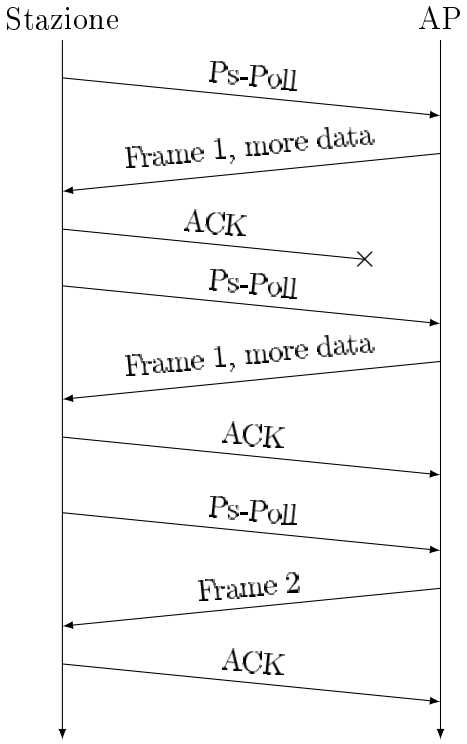


Figura 5.1: Sequenza di richiesta direcapito di frame tramite Ps-Poll.

Recapito dei frame Il processo di recapito dei frame in attesa nel buffer dell'AP è illustrato in figura 5.2 dove sono rappresentate due stazioni mobili in PSM associate allo stesso AP. Ad ogni Beacon Interval l'Access Point trasmette un Beacon frame contenente indicazioni di traffico TIM. La stazione 1 ha un Listen Interval pari a 2 Beacon Interval mentre la stazione 2 ha un Listen Interval di 3 Beacon Interval.

All'istante dell'invio del primo Beacon vi sono frame nel buffer dell'AP destinati alla stazione 1, e nessun frame per la stazione 2. La stazione 2 si sveglia per ricevere il Beacon e deducendo che non ha frame in attesa di recapito ritorna allo stato dormiente.

Al secondo Beacon il TIM indica che vi sono frame destinati sia alla stazione 1 che alla stazione 2 ma solo la stazione 1 si sveglia per ascoltare tale Beacon. Quest'ultima avvia la richiesta di recapito del frame in attesa inviando un Ps-Poll all'AP che risponderà inviando il frame nel quale il campo More Data non è posto ad 1. La procedura termina quindi con l'invio all'AP di un ACK al termine della ricezione del frame e con il ritorno della stazione 1 allo stato dormiente.

All'invio del terzo Beacon, il cui TIM indica frame per entrambe le stazioni, nessuna delle due stazioni si sveglia e l'AP dovrà quindi mantenere memorizzati nel buffer i frame in attesa di risvegli agli istanti di Beacon successivi.

[2]L'Association ID (AID) è un numero identificativo della stazione che viene inviato dall'Access Point non appena la procedura di associazione termina con esito positivo. Tale numero è utilizzato per identificare la stazione nel caso vi siano dei frame in attesa di recapito quando è attivata la modalità Power-saving.

Al momento dell'invio del quarto Beacon entrambe le stazioni hanno in programma di svegliarsi secondo il loro Listen Interval. Al loro risveglio entrambe vedono che vi è un frame in attesa e devono trasmettere un PS-Poll. Poiché ciascuna delle due stazioni grazie alla lettura del TIM è consapevole della presenza di almeno un'altra stazione in procinto di inviare un PS-Poll per ricevere frame in attesa, per evitare collisioni avvia un timer di backoff T_{BO} generato casualmente secondo la regola 5.1. Supponendo che il T_{BO} generato dalla stazione 1 sia il minore tra i due, questa invierà al termine del Tempo di backoff il PS-Poll all'AP mentre la stazione 2 rinvierà la trasmissione congelando il timer fino a quando il mezzo non tornerà libero. Al termine della procedura di trasferimento del frame alla stazione 1 e del riscontro ACK inviato da questa, la stazione 2 riprende il conto alla rovescia del T_{BO} precedentemente interrotto. Una stazione 3 (non indicata in figura) effettua una trasmissione[3] prima che il T_{BO} della stazione 2 termini occupando il canale e costringendo nuovamente la stazione 2 a rimandare la trasmissione del suo PS-Poll rimanendo in ascolto fino al TIM del successivo Beacon. Se nel frattempo il buffer dell'AP si è riempito ed è stato deciso di scartare il frame destinato alla stazione 2, il TIM del quinto Beacon non avrà più il bit relativo all'AID della stazione 2 contrassegnato ad 1 e pertanto questa potrà tornare in *sleep-mode*.

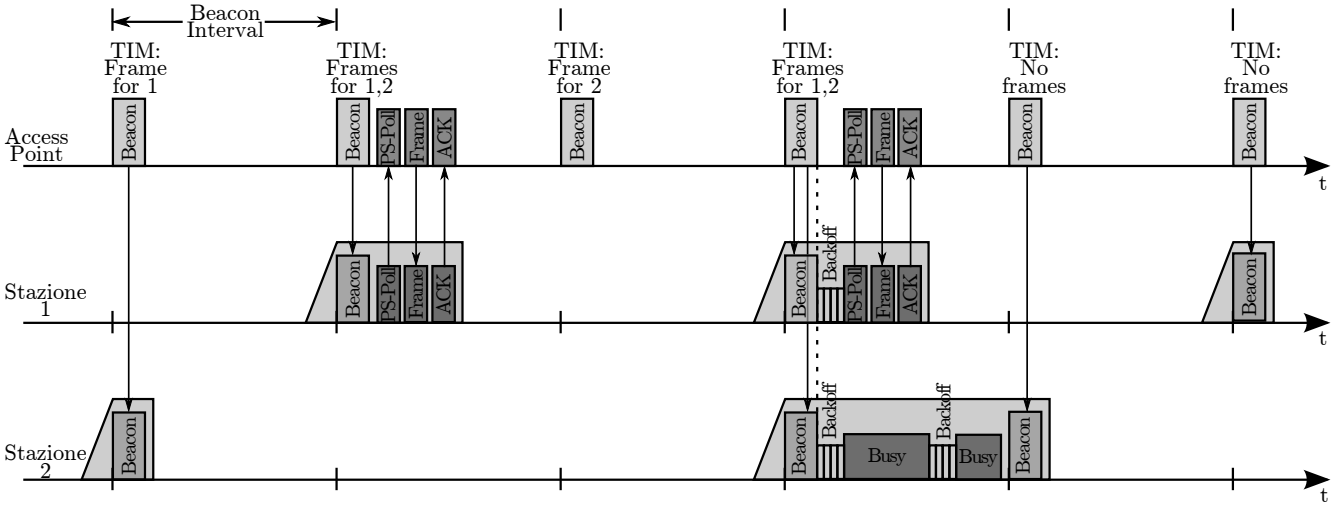


Figura 5.2: Procedura di recapito dei frame nel buffer dell'AP alle stazioni in PSM.

5.2 Consumi energetici

Sono presentati in questa sezione alcuni risultati sperimentali riguardanti i consumi energetici del modulo RTX4100 in vari scenari utilizzando il protocollo CoAP e il protocollo HTTP.

Come si nota dalla Applicarion Note *Current Consumption* fornita con la scheda RTX4100, i consumi energetici di una trasmissione continua di segmenti TCP o UDP non variano significamente impostando una potenza di trasmissione di 0 dBm (potenza minima) o 18 dBm (vicino al limite della normativa ETSI EN 301 893): si osserva un incremento di appena il 4.3 % della corrente media per una trasmissione UDP a 2,55 Mbit/s e del 2.1 % per una trasmissione TCP allo stesso rate e sempre usando lo standard 802.11n. Per rate molto più bassi non si notano differenze nei consumi pertanto la potenza di trasmission non risulta un fattore determinante per

[3] A differenza della ricezione di frame, lo standard 802.11 non vincola le stazioni in PSM a trasmettere un frame solo in corrispondenza degli istanti di Beacon: la stazione può passare dallo stato dormiente a quello attivo e trasmettere immediatamente il frame dopo una preliminare procedura DCF, ovvero ascolto del canale per un DIFS ed eventuale backoff nel caso di canale occupato.

le misure che si andranno a compiere qui di seguito dove i datagrammi o segmenti scambiati hanno un payload di poche decine di byte.

Tutte le misurazioni sono state effettuate connettendosi allo stesso Access Point 802.11n posto a distanza di 5 m impostando la stessa potenza di trasmissione massima a 1 mW (0 dBm), sicurezza WPA2 Pre-Shared Key e in presenza di basso carico di traffico.

5.2.1 Setup sperimentale

Per effettuare delle misure di corrente è stato sfruttato un resistore da 0.1 Ω (tolleranza dell'1 %) presente in serie nel cammino verso massa dal terminale VBAT− di alimentazione a batteria. Connettendo puntale e collegamento di massa di una sonda passiva 1x ai due terminali del resistore è stata prelevata la tensione ai capi di questo e visualizzata su un oscilloscopio cui è collegata la sonda.

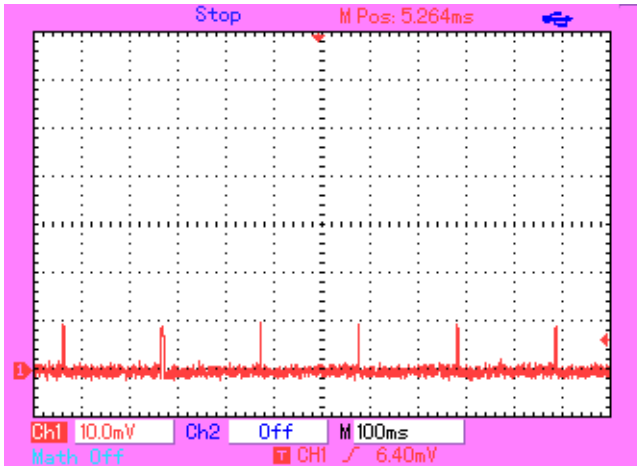
Le tensioni visualizzate possono essere immediatamente tradotte in corrente dal valore della resistenza: una tensione istantanea di 10 mV equivale ad una corrente istantanea di 100 mA.

Come indicato nella sezione 3.3 la corrente media misurata dovuta alla sola attività della CPU quando il chip Wi-Fi è spento è dell'ordine dei 2.7 μA ed è stata trascurata nelle misure effettuate non essendo visualizzabile con il metodo indicato a causa del basso valore di resistenza del resistore impiegato.

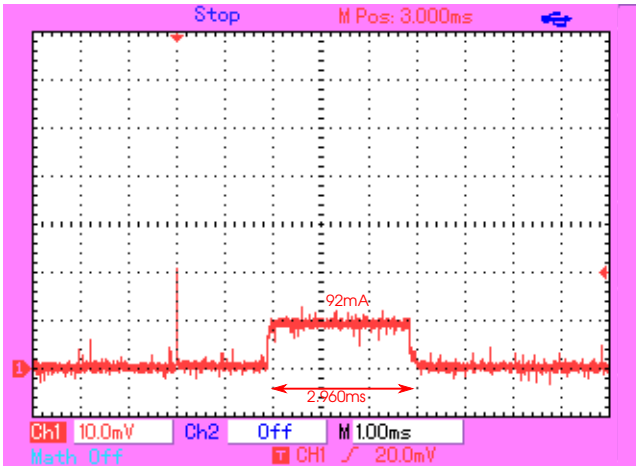
5.2.2 Ascolto dei Beacon

In figura 5.3(a) sono riportati i profili di corrente tipici di una stazione in PSM che si risveglia per ascoltare un beacon ogni 200 ms. Come si vede dal dettaglio di figura 5.3(b) in cui è visualizzato il profilo tipico di un singolo ascolto di un Beacon, la corrente erogata dalla batteria è approssimativamente costante e pari a 92 mA per 2.96 ms, pertanto la quantità di carica spesa per l'accensione del chip Wi-Fi e ascolto di un Beacon che non porti indicazione di traffico in attesa per la stazione in esame (ovvero che non provochi l'invio di un PS-Poll per ricevere i frame nel buffer) è pari a

$$Q_{\text{Beacon}} = 272.3\,\mu\text{C}\,.$$



(a) Ascolto di un Beacon ogni 200 ms



(b) Dettaglio di un singolo Beacon

Figura 5.3: Profili di corrente durante la fase di risveglio e ascolto del Beacon.

5.2.3 Dimensione del pacchetto

Nei test condotti si è limitata la dimensione dei datagrammi UDP e dei segmenti TCP in modo da evtare frammentazione IP, ovvero trasmettendo datagrammi UDP e segmenti TCP aventi payload non eccedenti MTU−28 byte e MTU−40 byte rispettivamente. Tale scenario è piuttosto comune nelle applicazioni IoT in esame come reti di sensori dove l'informazione da scambiare è una lettura di un dato acquisito da una periferica la cui dimensione è normalmente di pochi byte.

Per avere una idea di come agisce la frammentazione IP sui consumi energetici delle stazioni Wi-Fi in figura 5.4 è riportato il profilo di corrente di una ricezione di un pacchetto UDP avente payload di 500 byte mentre in figura 5.5 è riportato il profilo di corrente di una ricezione di un pacchetto UDP avente payload di 1500 byte, ovvero di un pacchetto IP di 1528 byte, di poco superiore all'MTU di 1500 byte dello strato IP nel collegamento tra modulo RTX4100 ed Access Point.

La carica spesa nella intera sequenza è facilmente calcolabile dalle figure 5.4 e 5.5 e pari, per la ricezione del pacchetto da 528 B a

$$Q^{(\mathrm{RX})}_{528\,\mathrm{B}} = 428.42\,\mu\mathrm{C}$$

e per la ricezione del pacchetto frammentato da 1528 B pari a

$$Q^{(\mathrm{RX})}_{1528\,\mathrm{B}} = 16.134\,\mathrm{mC}$$

La motivazione della maggiore lunghezza di tale sequenza di ricezione rispetto al caso del singolo PDU è insita nella politica di recapito dei frame alle stazioni in PSM da parte dell'Access Point. Un Access Point può infatti inviare immediatamente un frame alla stazione che lo ha richiesto tramite PS-Poll oppure può decidere di inviare semplicemente un ACK di ricezione del PS-Poll e ritardare la consegna (*deferred response*). Lo svantaggio per la stazione in PSM è che una volta inviato il PS-Poll e ricevuto l'ACK da parte dell'Access Point, quest'ultimo può inviare il frame richiesto in qualsiasi momento, costringendo la stazione ad attendere in Active Mode fino alla ricezione del frame o fino alla ricezione di un Beacon nella cui TIM il proprio bit di AID sia nullo.

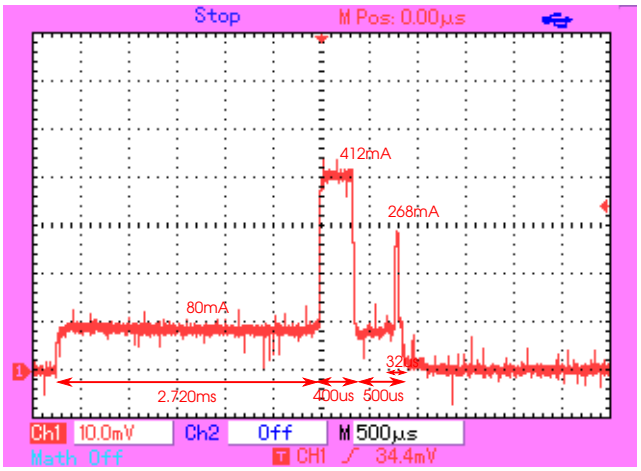
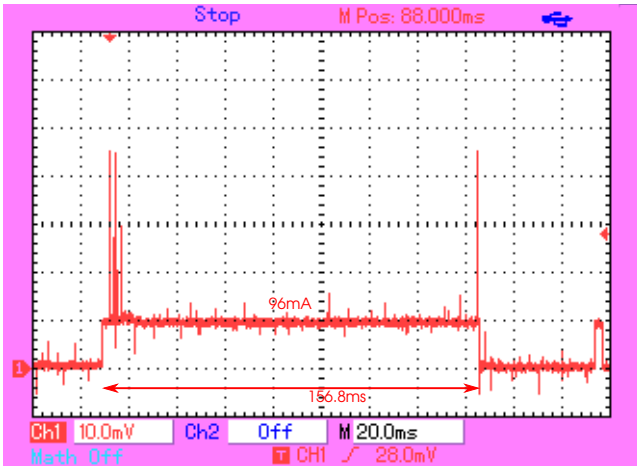


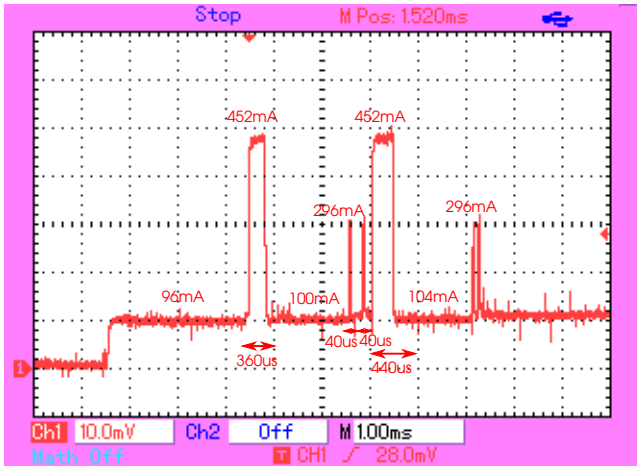
Figura 5.4: Profilo di corrente di una ricezione di un pacchetto UDP di dimensione 528 byte, header UDP e IP compresi.

5.2.4 Server CoAP - Risposta Piggy-backed

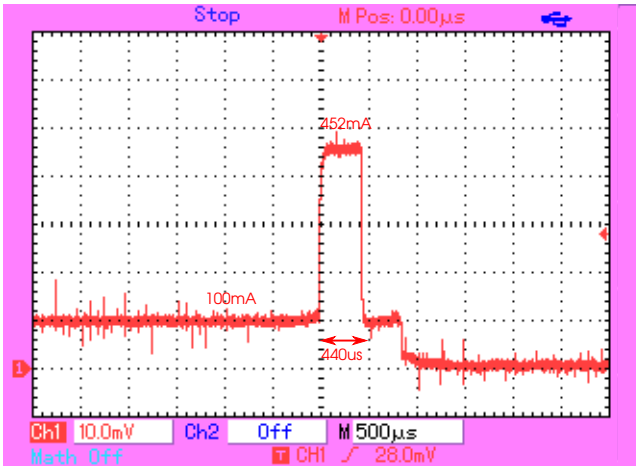
Come prima situazione di esempio si considera il caso di un server CoAP in PSM che realizza un nodo di monitoraggio della temperatura che è sempre disponibile a ricevere richieste di tipo



(a) Intera sequenza



(b) Dettaglio iniziale



(c) Dettaglio finale

Figura 5.5: Profilo di corrente di una ricezione di un pacchetto UDP di dimensione 1528 byte, header UDP e IP compresi.

GET e ad inviare la temperatura in una risposta che supponiamo *piggy-backed* (si suppone che la lettura della temperatura dal sensore avvenga sempre entro il tempo $T_{\text{app resp pb}}$).

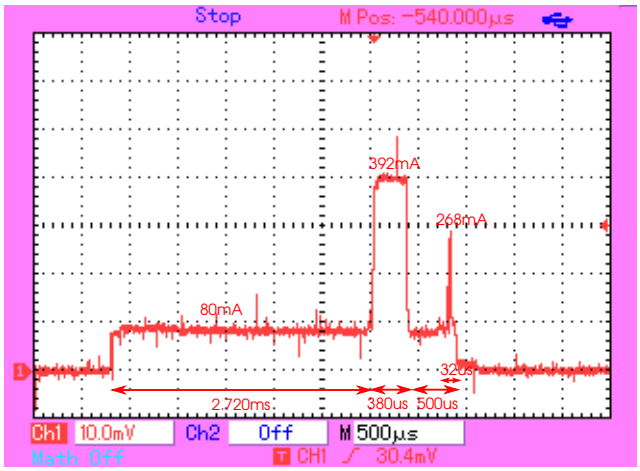
In figura 5.6 sono riportati due dettagli del profilo di corrente di una risposta CoAP piggy-backed: la ricezione del messaggio di richiesta (figura 5.6(a)) e l'invio della risposta allegata all'ACK di riscontro di ricezione della richiesta (figura 5.6(b)).

Il messaggio di richiesta GET ricevuto è composto di 16 byte a livello CoAP: 4 byte di header, 1 byte di token, 1 byte di intestazione del campo opzione Uri-Path e 11 byte del valore dell'opzione Uri-Path `temperature`. Il numero di byte totali del frame ricevuto è 58 bytes. La risposta è invece formata da 11 byte a livello CoAP: 4 byte di header, 1 byte per il payload marker e 6 byte per il payload che rappresenta la risorsa temperatura: `22.6 C`. Il frame inviato ha dimensione totale (livello DLL) di 53 byte.

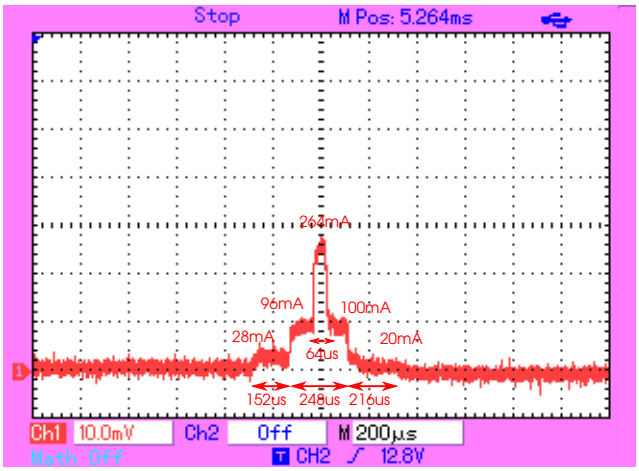
La prima cosa che si nota è che integrando i profili di corrente appena citati si ottiene che la carica spesa per la ricezione della richiesta, pari a 412.58 mC, risulta maggiore di quella spesa per la trasmissione del riscontro e risposta che è di appena 43.49 μC. La motivazione dietro a questo fatto è da ricercare nella modalità di recapito dei frame destinati alle stazioni in PSM. Come illustrato nella sezione 5.1 una stazione operante in PSM deve ascoltare periodicamente il Beacon trasmesso dall'Access Point. Riosservando la figura 5.3(b) si nota come i primi 2.9 ms del profilo di ricezione 5.6(a) siano sovrapponibili al profilo di ascolto del beacon. Una stazione che

voglia ricevere frame in attesa nel buffer dell'AP oltre ad ascoltare il Beacon deve effettuare una trasmissione di un PS-Poll (con eventuale attesa nel caso di canale occupato), attendere l'arrivo del frame mantenendo il modulo Wi-Fi attivo, ricevere il frame ed inviare un frame ACK di riscontro all'AP. La ricezione si compone pertanto di più operazioni che la rendono dispendiosa in termini energetici se la si confronta con la trasmissione, la quale può avvenire in qualsiasi momento semplicemente riattivando il chip Wi-Fi e trasmettendo il frame dopo aver verificato per pochi μs che il canale sia libero. Chiaramente nel caso di traffico intenso verrà avviata la procedura DCF di backoff esponenziale e il modulo Wi-Fi dovrà rimanere attivo ed in ascolto del canale fino a quando questo non sia libero.

I bassi consumi appena indicati (456 μC) per la ricezione di una richiesta e invio della risposta tuttavia sono trascurabili se si confrontano con i 272.3 μC spesi ogni 100 ms per la ricezione del Beacon. Una modalità più efficiente di realizzare un sensore di temperatura è in modalità client, come si vedrà nella sezione 5.2.6, con sospensione per lunghi periodi dell'attività Wi-Fi.



(a) Ricezione della richiesta



(b) Invio dell'ACK con risposta piggy-backed di dimensione 10bytes (livello CoAP)

Figura 5.6: Profilo di corrente di una risposta CoAP piggy-backed nell'ACK.

5.2.5 Server HTTP

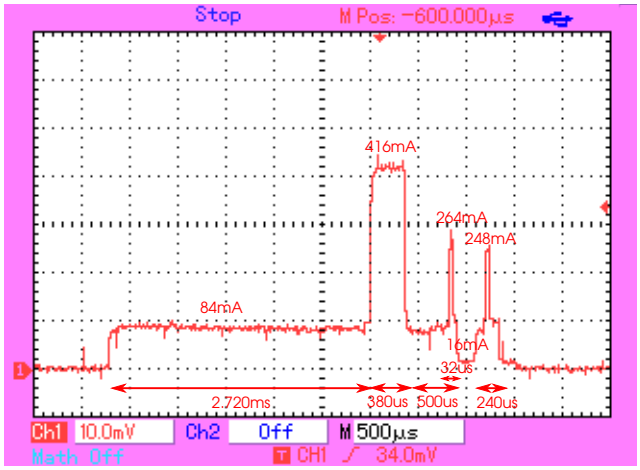
Lo stesso scenario descritto nel caso del Server CoAP può essere realizzato anche impiegando il protocollo HTTP. In figura 5.7 sono rappresentati due dettagli del profilo di corrente di una sessione server HTTP al momento dell'arrivo di una richiesta remota: la ricezione della richiesta con immediato invio dell'ACK di riscontro (figura 5.7(a)) e la ricezione dell'ACK dal client che conferma la ricezione della risposta. Non è riportato per brevità il profilo della trasmissione della risposta.

I consumi di carica sono pari a 434.32 μC e 26.43 μC per la ricezione della richiesta e invio del riscontro ACK (figura 5.7(a)), 43.587 μC per la trasmissione della risposta e 413.824 μC per la ricezione dell'ACK alla risposta (5.7(b)).

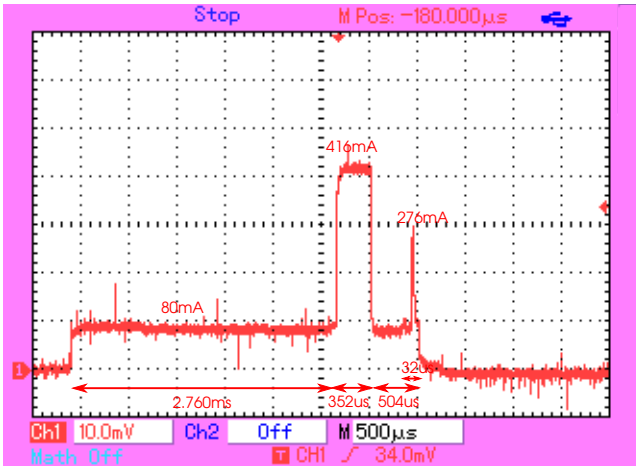
Anche in questo caso un modo più efficiente di implementare un sensore di temperatura riducendo i consumi derivanti dalla ricezione periodica dei Beacon è tramite client con sospensione del Wi-Fi come si vedrà in sezione 5.2.7.

5.2.6 Client CoAP

Si illustrano ora i profili di consumo ottenuti da misure effettuate sul modulo RTX4100 in modalità client CoAP impiegato come sensore che riporta la temperatura misurata ad un server CoAP



(a) Ricezione della richiesta e immediato invio dell'ACK



(b) ricezione dell'ACK dal client

Figura 5.7: Profilo di corrente di una risposta HTTP.

tramite una richiesta PUT che si occupa di immagazzinare tale informazione per elaborazione o per renderla disponibile ad altri nodi che vogliano conoscere questa informazione.

Per una applicazione di monitoraggio di temperatura ambientale normalmente è richiesta una misura di temperatura ogni 5 minuti. Il client dunque effettua una breve trasmissione con riscontro all'interno di un lungo periodo di inattività in cui viene spesa energia per ricevere i Beacon dall'Access Point. Con lo standard Power-Saving mode 802.11 difficilmente è possibile impostare un Listen Interval dell'ordine dei minuti in quanto l'Access Point rischia di non avere sufficiente spazio nel suo buffer per immagazzinare eventuali frame destinati alla stazione che giungano in tale intervallo e rifiuta la richiesta imponendo un Listen Interval standard dell'ordine dei 100, 200 ms. Per ridurre i consumi energetici tra una trasmissione di temperatura e un altra è possibile sospendere il chip Wi-Fi del client al termine di ogni trasmissione di temperatura senza disconnettersi dall'Access Point per poi riattivarlo subito prima della successiva trasmissione.

Il profilo di corrente tipico di una sequenza di riattivazione del modulo Wi-Fi dopo un periodo di sospensione, trasmissione di una richiesta PUT, attesa del riscontro ACK e sospensione del chip Wi-Fi è illustrato in figura 5.8. Come si vede esso è composto da

- una sequenza iniziale di riattivazione del chip Wi-Fi formata dal profilo riportato parzialmente in figura 5.10(a) (non è visualizzata una componente costante iniziale di 184 ms che è riportata invece in figura 5.8) e dall'ascolto di 9 Beacon, che consuma

$$\begin{aligned} Q_{\text{resume}} &= 184\,\text{ms} \cdot 32\,\text{mA} + 66.8\,\text{ms} \cdot 116\,\text{mA} \\ &\quad + 2 \cdot 400\,\mu\text{s} \cdot (488 - 116)\,\text{mA} + 440\,\mu\text{s} \cdot (488 - 116)\,\text{mA} \\ &\quad + 9\,Q_{\text{Beacon}} \\ &= 14.01\,\text{mC} + 2.45\,\text{mC} \\ &= 16.46\,\text{mC}\,; \end{aligned} \tag{5.2}$$

- la trasmissione della richiesta PUT e la ricezione dell'ACK *piggy-backed* inframezzate dall'ascolto di 2 beacon:

$$\begin{aligned} Q_{\text{Tx,Rx}}^{\text{CoAP,Client}} &= Q_{\text{Tx PUT}}^{\text{CoAP}} + 2\,Q_{\text{Beacon}} + Q_{\text{Rx ACK}}^{\text{CoAP}} \\ &= 41.408\,\mu\text{C} + 547.2\,\mu\text{C} + 675.52\,\mu\text{C} \\ &= 1.264\,\text{mC}\,; \end{aligned}$$

dove $Q_{\text{Tx PUT}}^{\text{CoAP}}$ è la carica spesa dal client per la trasmissione del messaggio CoAP di richiesta PUT, il cui profilo è riportato in figura 5.9(a), e $Q_{\text{Rx ACK}}^{\text{CoAP}}$ è l'energia spesa per la ricezione del

riscontro vuoto alla richiesta CON inviata (4 byte a livello CoAP,), il cui profilo è riportato in figura 5.9(b);

- la sequenza di sospensione del chip Wi-Fi, formata da 4 ascolti di Beacon e da un profilo finale riportato in figura 5.10(b):

$$\begin{aligned} Q_{\text{suspend}} &= 4\, Q_{\text{Beacon}} + 2.2\,\text{ms} \cdot 112\,\text{mA} + 2 \cdot 400\,\mu\text{s} \cdot (484 - 112)\,\text{mA} \\ &= 1.633\,\text{mC}\,; \end{aligned} \tag{5.3}$$

La carica totale spesa dunque in una sequenza di riattivazione, invio di richiesta PUT, ricezione di ACK vuoto e sospensione risulta

$$\begin{aligned} Q_{\text{report}}^{\text{CoAP,Client}} &= Q_{\text{resume}} + Q_{\text{Tx,Rx}}^{\text{CoAP,Client}} + Q_{\text{suspend}} \\ &= 19.357\,\text{mC}\,; \end{aligned} \tag{5.4}$$

Si noti che in una applicazione di tipo client i timer T_{dup} e $T_{\text{cache rtx}}$ sono stati ridotti a 100 ms in modo che non appena avviene la ricezione dell'ACK o della risposta CON *separate* il client avvii subito la sospensione del chip Wi-Fi per risparmiare energia evitando così di ricevere ulteriori Beacon.

Il numero di intervalli di Beacon che richiedono le sequenze di riattivazione e sospensione sono determinati dalla specifica implementazione di tali procedure all'interno della scheda. Esse vengono semplicemente osservate e riportate poiché tali sequenze saranno tipicamente ripetute con le stesse tempistiche per una applicazione client HTTP che è riportata nella sezione successiva.

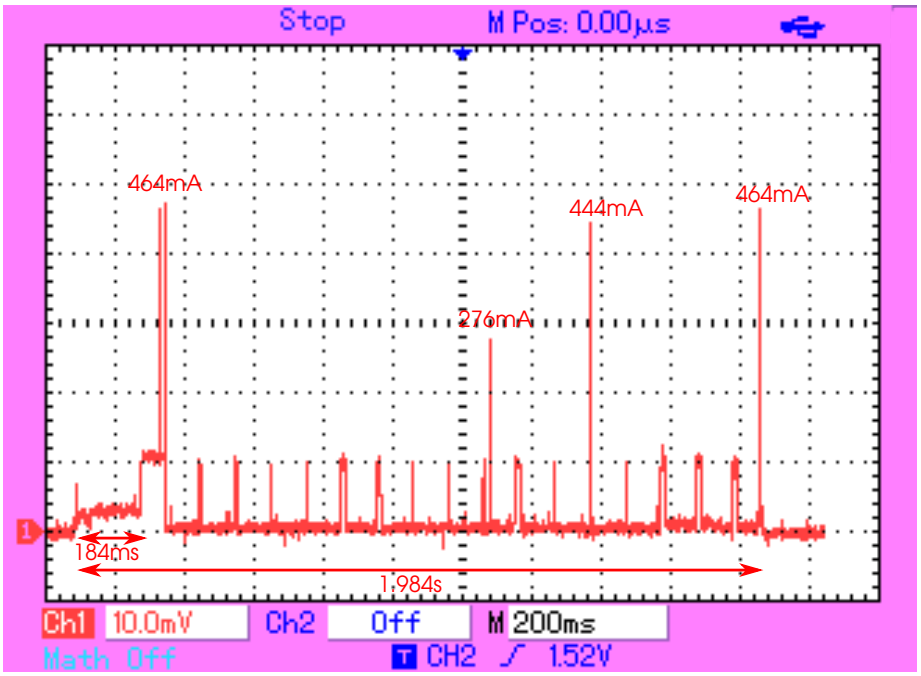
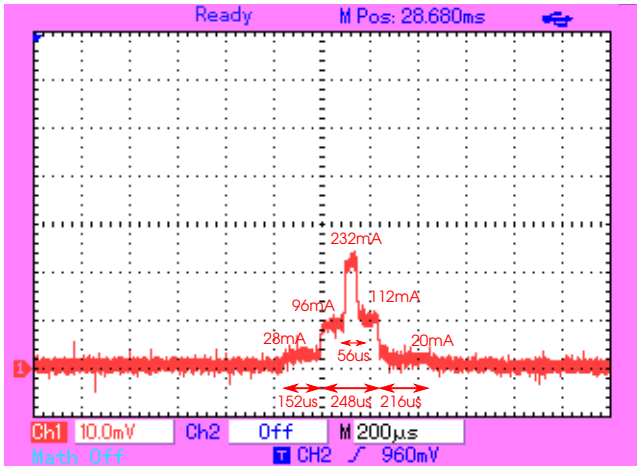


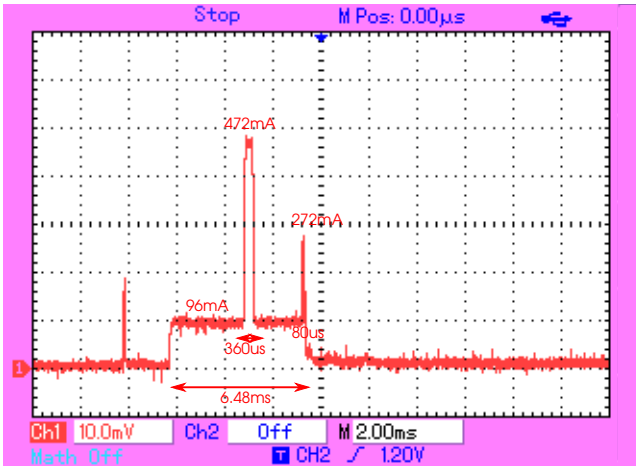
Figura 5.8: Profilo di corrente della sequenza di riattivazione, trasmissione di una richiesta PUT, attesa del riscontro ACK e sospensione del chip Wi-Fi.

5.2.7 Client HTTP

É stato riprodotto lo stesso scenario del Client CoAP utilizzando per la richiesta di PUT il client HTTP implementato tramite le API fornite dal firmware. Si noti che le API HTTP fornite non

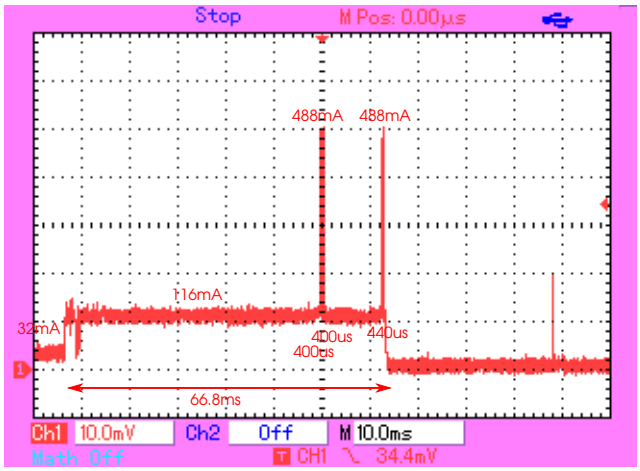


(a) Trasmissione della richiesta PUT

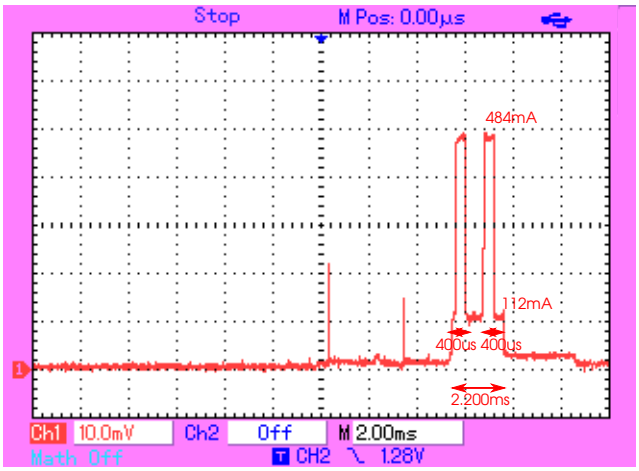


(b) Ricezione del riscontro ACK vuoto

Figura 5.9: Dettagli della trasmissione della richiesta PUT, e della ricezione del riscontro ACK vuoto.



(a) Dettaglio iniziale di riattivazione



(b) Dettaglio finale di sospensione

Figura 5.10: Dettagli iniziale e finale della sequenza di riattivazione e sospensione del chip Wi-Fi.

permettono di allegare messaggi di richiesta/risposta agli ACK TCP e dunque queste saranno sempre inviate in segmenti separati dai riscontri.

Nel caso di un client HTTP che invia richieste di *report* della temperatura ad intervalli dell'ordine dei minuti è bene considerare separatamente i casi in cui la sessione HTTP persistente rimanga ancora aperta e quello in cui possa essere chiusa dal server obbligando il client ad effettuare nuovamente l'handshake. In questo secondo caso all'invio della richiesta PUT il server risponderà con un segmento TCP di RST obbligando il client ad effettuare l'handshake e a ritrasmettere la richiesta PUT.

Sessione aperta Nel caso ottimo in cui la sessione TCP sia rimasta attiva la sequenza delle operazioni che il client effettua per eseguire il report della temperatura al server è la seguente:

- sequenza di riattivazione del modulo Wi-Fi come indicato nella (5.2)

$$Q_{\text{resume}} = 16.46\,\text{mC}\,;$$

- trasmissione della richiesta PUT di 78 byte a livello HTTP, ovvero di un frame di 132 byte, che provoca un dispendio di carica

$$Q^{\text{HTTP,Client}}_{\text{Tx PUT}} = 45.894\,\mu\text{C}\,; \tag{5.5}$$

- ricezione di un ACK TCP di 54 byte a livello DLL che provoca un dispendio di carica

$$Q^{\text{HTTP,Client}}_{\text{Rx TCP ACK}} = 434.32\,\mu\text{C}\,;$$

- ricezione della risposta HTTP alla richiesta PUT che comporta un dispendio di carica analogo a quello dell'ACK TCP appena ricevuto

$$Q^{\text{HTTP,Client}}_{\text{Rx OK}} \simeq Q^{\text{HTTP,Client}}_{\text{Rx TCP ACK}}$$

- invio dell'ACK TCP (54 byte a livello DLL) della risposta appena ricevuta

$$Q^{\text{HTTP,Client}}_{\text{Tx TCP ACK}} = 43.285\,\mu\text{C}\,;$$

- sequenza di sospensione del modulo Wi-Fi come indicato nella (5.3)

$$Q_{\text{suspend}} = 1.633\,\text{mC}\,.$$

Ai valori di carica indicati vanno aggiunti inoltre i consumi dovuti all'ascolto dei beacon che recanti indicazione di traffico giunti all'interno dell'intervallo di tempo di durata della sequenza, pari a $525,12\,\text{ms}$[4]:

$$Q^{\text{HTTP,Client}}_{\text{Beacons}} = 3\;Q_{\text{Beacon}} = 816.9\,\mu\text{C}\,.$$

La carica totale spesa dal client per un report di temperatura si ottiene sommando tutti gli addendi appena indicati:

$$Q^{\text{HTTP,Client}}_{\text{report,open}} = 19.868\,\text{mC}\,. \tag{5.6}$$

Sessione chiusa Nel caso in cui la sessione TCP sia stata terminata dal server, alle operazioni indicate nel paragrafo precedente vanno aggiunte tra la fine della sequenza di riattivazione e l'invio della richiesta PUT i seguenti eventi:

- invio di una prima richiesta PUT uguale alla (5.5) ma che non provocherà l'ottenimento della risposta bensì la ricezione di un RST TCP dal server che ha terminato la sessione;
- ricezione del segmento TCP di RST (54 byte a livello DLL) che provoca un dispendio di

$$Q^{\text{HTTP,Client}}_{\text{Rx TCP RST}} = 434.32\,\mu\text{C}\,;$$

- l'avvio della procedura di handshake comprendente
  - l'invio di un segmento SYN (66 byte a livello DLL) che consuma $43.748\,\mu\text{C}$;
  - la ricezione di un segmento SYNACK (avente i campi SYN e ACK a 1) di 58 byte che consuma circa quanto un ACK TCP:

$$Q^{\text{HTTP,Client}}_{\text{Rx SYNACK}} \simeq Q^{\text{HTTP,Client}}_{\text{Rx TCP ACK}}\,;$$

[4]Si noti che a differenza del caso client CoAP è stato usato qui un server HTTP non in PSM che ha una latenza minore potendo ricevere segmenti TCP in qualsiasi momento e non soltanto in corrispondenza dei beacon periodici di periodo 100 ms. Questo ha generato la ricezione di meno Beacon da parte del client.

– l'invio di un ACK che consuma

$$Q^{\text{HTTP,Client}}_{\text{Tx TCP ACK}} = 43.285\,\mu\text{C}\,;$$

A tal punto il client può inviare la richiesta PUT seguendo le operazioni descritte nel paragrafo precedente. Vanno contati anche qui i dispendi di carica dovuti alle ricezioni di Beacon che non trasportano indicazioni di traffico destinato al client che ammontano ad altri 2 Beacon ricevuti, ovvero ad ulteriori

$$2\,Q_{\text{Beacon}} = 544.6\,\mu\text{C}\,.$$

Sommando i consumi appena descritti a quelli $Q^{\text{HTTP,Client}}_{\text{report,open}}$ del paragrafo precedente si ottiene

$$Q^{\text{HTTP,Client}}_{\text{report,closed}} = 21.414\,\text{mC}\,. \tag{5.7}$$

5.2.8 Corrente media

É possibile calcolare ora la corrente media erogata dalla batteria negli scenari client CoAP e client HTTP appena descritti nelle sezioni 5.2.6 e 5.2.7. La corrente media che l'alimentazione a batteria fornisce nei tre scenari considerati si calcola

$$I_{\text{avg}} = \frac{I_{\text{sleep}}(T_{\text{report}} - T_{\text{active}}) + Q^{\text{Client}}_{\text{report}}}{T_{\text{report}}} \tag{5.8}$$

dove $Q^{\text{Client}}_{\text{report}}$ è la carica (5.4), (5.6) o (5.7) spesa dal client CoAP, HTTP senza handshake ed HTTP con handshake rispettivamente per l'intera sequenza di report; T_{active} è il tempo impiegato per la sequenza di report; T_{report} è il periodo di report scelto, ovvero l'intervallo di tempo che si sceglie di impostare tra due report successivi; e $I_{\text{sleep}} = 2.7\,\mu\text{A}$ è la corrente media dovuta al solo contributo del MCU durante i periodi di sospensione quando il chip Wi-Fi è sospeso. Nella tabella 5.2 sono riassunti i parametri per il calcolo della (5.8).

	Client CoAP	**Client HTTP senza handshake**	**Client HTTP con handshake**
$Q^{\text{Client}}_{\text{report}}$	19.357 mC	19.868 mC	21.414 mC
T_{active}	1.984 s	2.225	2.657 s

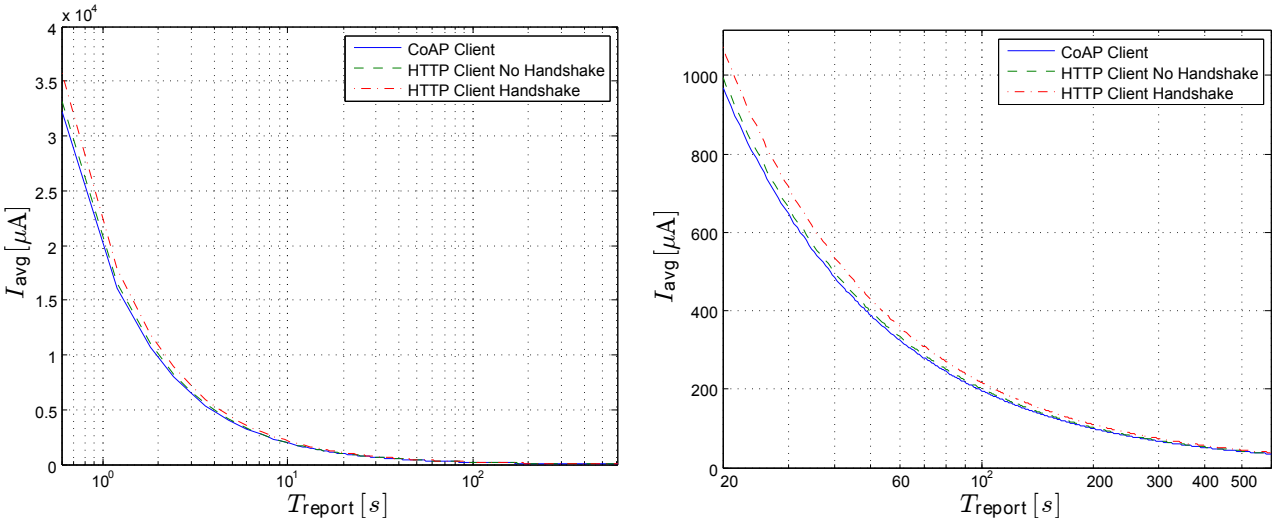
Tabella 5.2: Parametri dei client CoAP e HTTP per il calcolo della corrente media I_{avg}.

Rappresentando la (5.8) in funzione del tempo di report si ottengono le curve di figura 5.11.

Si riportano in tabella alcuni valori della (5.8) per T_{report} che varia da 1 a 5 minuti. In tabella 5.4 sono indicati i risparmi di corrente I_{avg} utilizzando il protocollo CoAP rispetto a quello HTTP nei due casi di assenza e presenza di handshake per $T_{\text{report}} = 1 \div 5$ minuti. Il risparmio, decrescente all'aumentare di T_{report}, che si ottiene è pari circa al 2.5% rispetto al protocollo HTTP senza handshake e circa 9.5% rispetto al protocollo HTTP con handshake.

	60 s	**120 s**	**180 s**	**240 s**	**300 s**
CoAP Client	325.2 μA	164 μA	110.2 μA	83.33 μA	67.21 μA
HTTP Client senza Handshake	333.7 μA	168.2 μA	113 μA	85.46 μA	68.91 μA
HTTP Client senza Handshake	359.5 μA	181.1 μA	121.6 μA	91.9 μA	74.06 μA

Tabella 5.3: Alcuni valori della I_{avg}.



(a) Corrente I_{avg} per $T_{\text{report}} = 1 \div 500\,\text{s}$

(b) Dettaglio per $T_{\text{report}} = 20 \div 500\,\text{s}$

Figura 5.11: Corrente I_{avg} per client CoAP e HTTP in funzione del T_{report}.

	60 s	**120 s**	**180 s**	**240 s**	**300 s**
HTTP Client senza Handshake	2.55%	2.53%	2.51%	2.49%	2.47%
HTTP Client senza Handshake	9.5%	9.44%	9.38%	9.33%	9.25%

Tabella 5.4: Risparmio di corrente I_{avg} utilizzando il protocollo CoAP rispetto a quello HTTP nei due casi di assenza e presenza di handshake.

Capitolo 6

Conclusioni e sviluppi futuri

Si è visto come l'adozione di un protocollo CoAP rispetto ad HTTP consente di una modesta riduzione dei consumi nelle applicazioni di reti di sensori mantenendo affidabile lo scambio di messaggi. Dai risultati sperimentali emerge tuttavia che la componente dominante dei consumi energetici sia insita nella gestione del risparmio energetico dello standard 802.11 *Power-Saving Mode*. Uno standard PSM dove gli intervalli di Beacon fossero modificabili dinamicamente consentirebbe un risparmio di energia spesa per l'ascolto dei Beacon e eliminerebbe la necessità di eseguire una sequenza di sospensione e riattivazione del modulo Wi-Fi.

Lo standard WiFi 802.11 attuale pone dunque delle limitazioni all'utilizzo del Wi-Fi in scenari di interazione Macchina-Macchina (M2M), in particolare per reti di sensori. Le motivazioni principali sono le seguenti:

- la modalità Power-Saving è rigida in quanto permette di impostare permette di impostare Listen Interval dell'ordine di pochi Beacon;
- le bande Wi-Fi attuali (2.4 e 5.4 GHz) hanno un raggio di copertura molto corto e perdite consistenti nell'attraversamento di comuni ostacoli architetturali; ciò rende necessario l'utilizzo di nodi intermedi che aggiungono complessità alla rete quando, dati i bassi data rate richiesti da applicazioni M2M, sarebbe sufficiente impiegare delle bande con frequenze portanti più basse con rate più basso e maggiore capacità di penetrazione.

Per ovviare a queste limitazioni nel 2010 è stato creato lo IEEE Task Group 802.11ah (TGah) avente come obiettivo la creazione di 802.11ah: uno standard globale Sub-1 GHz WLAN per applicazioni a comunicazioni M2M con un gran numero di nodi, un raggio di copertura ampio ed autonomie estese dei singoli dispositivi.

Gli obiettivi di design di questo nuovo standard sono:

- supporto fino a 8 191 stazioni associate ad un unico AP attraverso una struttura gerarchica di ID;
- implementazione di strategie di risparmio energetico;
- data rate minimo della rete di 100 kbit/s;
- frequenze delle portanti attorno ai 900 MHz;
- raggio di copertura dell'Access Point fino ad 1 km all'esterno;
- topologia della rete a stella (single-hop);
- trasmissioni di dati brevi e poco frequenti;

- alta affidabilità grazie alla scarsa congestione e alla minore attenuazione della banda Sub-1 GHz impiegata.

In particolare 802.11ah offre alle stazioni la possibilità di essere associate in tre modalità:

TIM Station : questo è l'unico tipo di stazione che deve ascoltare i Beacon dall'AP per ricevere o mandare frame; la trasmissione si effettua entro un periodo chiamato finestra di accesso ristretta (RAW) per tre tipi di segmenti (Multicast, Downink e Uplink); Questo tipo di stazione è indicato per dispositivi con un alto traffico;

Non-TIM Station questo tipo di stazione non deve ascoltare i Beacon periodici per trasmettere dati ma può trasmettere in una finestra di accesso ristretta periodica (PRAW) negoziata con l'AP durante l'associazione;

Unscheduled Station nemmeno le stazioni unscheduled devono ascoltare i Beacon e possono mandare un frame di poll per richiedere accesso al canale all'AP che risponderà con l'indicazione di un intervallo al di fuori delle RAW e PRAW durante il quale la stazione ha il permesso di accedere al canale.

Tutte le stazioni TIM, Non-TIM e Unscheduled possono impostare periodi di riposo molto lunghi, fino all'ordine degli anni, estendendo alcuni dei parametri di gestione del PSM 802.11 in fase di associazione con l'AP. Tuttavia il rischio di perdita della sincronizzazione con l'AP con intervalli di riposo così lunghi è un fattore da tenere in considerazione.

Bibliografia

[1] T. Berners-Lee, R. Fielding, and L. Masinter. RFC 3986, Uniform Resource Identifier (URI): Generic Syntax. http://rfc.net/rfc3986.html, 2005.

[2] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying Event-driven Programming of Memory-constrained Embedded Systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. http://tools.ietf.org/html/rfc2616, 1999.

[4] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Phd thesis, University of California, 2000.

[5] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders, 1st edition edition, March 2006.

[6] IEEE Standard for Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, IEEE Std 802.11-2012, March 2012.

[7] J. Postel. RFC 768: User datagram protocol. http://tools.ietf.org/html/rfc768, August 1980.

[8] J. Postel. RFC 793: Transmission control protocol. http://tools.ietf.org/html/rfc793, September 1981.

[9] S. Tozlu, M. Senel, Wei Mao and A. Keshavarzian. Wi-Fi enabled sensors for internet of things: A practical approach. *IEEE Communications Magazine*, 50(6):134–143, June 2012.

[10] Martin S. Nicklous Thomas Stober Uwe Hansmann, Lothar Merk. *Pervasive Computing: The Mobile World*. Springer-Verlag, 2nd edition edition, August 2003.

[11] C. Bormann Z. Shelby, K. Hartke. Constrained Application Protocol (CoAP). http://tools.ietf.org/search/draft-ietf-core-coap-18, 2013.