

# POLITECNICO DI MILANO

Facoltà di ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica



## Sincronizzazione low-power per reti di sensori wireless tramite controllo lineare decentralizzato: il protocollo FLOPSYNC

**RELATORE**

Chiar.mo Prof. Alberto LEVA

**CORRELATORE**

Ing. Federico TERRANEO

**TESI DI LAUREA DI**

**Luigi RINALDI**

**Matr. 783404**

**Anno Accademico 2012/2013**



*IERI TI HO GUARDATO*

*Per la prima volta ieri ti ho guardato  
con gli occhi del mio tempo più sereno.*

*Eri davanti a me, non t'ho sognato.*

*E la tua voce quasi, la sentivo  
« tranquillo Enri'... che io non t'ho lasciato».*

*Sentivo così forte il tuo pensiero  
ed eri dentro me così presente,  
che col pensiero tuo io c'ho parlato.  
Papà, oggi la pietà che m'hai insegnato  
mi rende più leggera la tua assenza  
e nel frattempo un tempo nuovo nasce:  
la cognizione di quello che sei stato.*

*L'immensa tenerezza in mi ti vivo  
e il nuovo padre che sei diventato.  
Oggi ricordarti non mi affligge più.  
E col tuo aiuto invento la mia vita,  
a volte mi diverto e faccio divertire.  
A volte è un vuoto che non so riempire.*

*Io col tuo aiuto canto la mia vita,  
dove ogni giorno vivi pure tu.  
E col tuo aiuto capisco un po' di più.*

*Capisco che il tuo ricordo è amore  
e la memoria la pago col dolore.*

*La lezione del tempo mi è servita,  
oggi ho capito perché... la vera vita  
finisce sempre e non è mai finita*

Enrico Brignano



# Ringraziamenti

In primo luogo desidero ringraziare il Prof. Alberto Leva per aver reso possibile lo sviluppo di questo lavoro e per avermi fatto crescere professionalmente.

Poi, ringrazio sentitamente l'Ing. Federico Terraneo correlatore di questo lavoro nonché meno amico, sempre pronto a dissipare i miei dubbi.

Ringrazio tutti gli amici di sempre, Christian, Elena, Nazario, Matteo, Lidia, Fabio con cui ho condiviso mille avventure e disavventure durante questi anni in quel di Milano e per essermi sempre stati vicini in ogni momento ed aver riposto in me la loro fiducia.

Un pensiero va anche a tutti i compagni conosciuti in questi anni di università, in particolare Giorgio, per l'aiuto e supporto nell'ultimo periodo.

Per ultima, ma non per questo meno importante, desidero ringraziare le persone che più di altre avrebbero meritato che questo lavoro fosse loro dedicato, la mia *famiglia*. In particolare ringrazio di cuore mia mamma, mio fratello Francesco, mia sorella Carmela e le mie due grandi gioie: il piccolo Gabriele e l'ultimo arrivato, Samuele. Ad essi va tutta la mia stima, il rispetto e la riconoscenza infinita, nonostante gli screzi e le incomprensioni che inevitabilmente si sono presentati nel corso degli anni. Impossibile poi dimenticarsi degli zii Antonio e Angela, i quali hanno avuto un importante ruolo nella mia educazione e crescita prodigandosi in ogni modo e maniera in loro possesso. Un ringraziamento va inoltre ai miei cugini Francesco e Giuseppe.



# Sommario

La sincronizzazione temporale è importante per il corretto funzionamento di molte applicazioni per reti di sensori wireless (WSN). Per esempio, applicazioni transazionali hanno bisogno di nodi con clock monotoni per evitare che eventi subiscano inversioni temporali, sistemi di gestione del power saving richiedono un overhead minimo di sincronizzazione per consentire il funzionamento a basso duty-cycle, applicazioni che affrontano ambienti estremi devono essere in grado di mantenere la precisione anche in presenza di variazioni termiche impreviste, e così via.

In questa tesi si propone uno schema di sincronizzazione del clock con compensazione di skew/drift per affrontare tutte le questioni sopra accennate. In concreto, la proposta consiste in un protocollo decentralizzato, indipendente dall'hardware, configurabile off-line, in grado di rigettare i disturbi termici, capace di mantenere la monotonia del clock locale ed allo stesso tempo dal basso impatto sulle risorse. Tutto questo è FLOPSYNC, acronimo di Feedback LOW Power SYNChronization, ideato seguendo una metodologia innovativa di progetto che vede la consolidata teoria del controllo applicata ai sistemi informatici.

Si propongono inoltre alcuni studi basati su esperimenti, allo scopo di mostrare la semplicità e l'efficacia del protocollo proposto.





# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Definizione di una WSN . . . . .	2
1.2	Scenari applicativi . . . . .	2
1.3	La sincronizzazione nel dominio WSN . . . . .	4
1.4	Obiettivi e contenuto . . . . .	5
<b>2</b>	<b>Stato dell'arte</b>	<b>7</b>
2.1	Oscillatore al quarzo . . . . .	7
2.1.1	Terminologia tecnica . . . . .	7
2.1.2	Frequenza di oscillazione . . . . .	11
2.2	Notazione e terminologia . . . . .	14
2.3	Tassonomia degli schemi di sincronizzazione . . . . .	16
2.4	Rassegna della letteratura . . . . .	20
2.4.1	Reference Broadcast Synchronization: RBS . . . . .	20
2.4.2	Timing-sync Protocol for Sensor Network: TPSN . . . . .	21
2.4.3	Delay Measurement Time Synchronization for WSNs: DMTS . . . . .	21
2.4.4	Flooding Synchronization Time Protocol: FTSP . . . . .	21
2.4.5	Lightweight Tree-based Synchronization: LTS . . . . .	22
2.4.6	Tiny-Sync e Mini-Sync (TS/MS) . . . . .	23
2.4.7	Scalable Lightweight Time Synchronization Protocol for WSNs: SLTP . . . . .	24
2.4.8	Reference Based, Tree Structured Time Synchronization: TSRT 25	
2.4.9	Feedback Based Synchronization: FBS . . . . .	25
2.5	Valutazione dei protocolli di sincronizzazione . . . . .	26
2.5.1	Valutazione hardware-dependent . . . . .	27
2.5.2	Valutazione hardware-independent . . . . .	28
2.6	Motivazione della ricerca . . . . .	29

<b>3</b>	<b>FLOPSYNC</b>	<b>31</b>
3.1	Approccio proposto . . . . .	31
3.2	La sincronizzazione come problema di controllo . . . . .	32
3.2.1	Modello del sistema da controllare . . . . .	33
3.2.2	Analisi del disturbo e modellazione . . . . .	35
3.2.3	Sintesi del controllore . . . . .	36
3.2.4	Criticità dei disturbi di origine termica . . . . .	39
3.2.5	Limitazione dell'errore e taratura del controllore . . . . .	42
3.3	Dal problema di controllo allo schema di sincronizzazione . . . . .	46
3.3.1	Un clock virtuale monotono . . . . .	49
3.4	Il protocollo di sincronizzazione FLOPSYNC . . . . .	51
3.4.1	Schema di flooding . . . . .	52
3.4.2	Controllore . . . . .	54
3.4.3	Clock virtuale . . . . .	55
3.4.4	Schema di re-sincronizzazione . . . . .	55
<b>4</b>	<b>Implementazione</b>	<b>57</b>
4.1	Tool di configurazione per FLOPSYNC . . . . .	57
4.2	Architettura hardware-software di un mote . . . . .	58
4.3	Il livello MAC dello stack radio . . . . .	62
4.4	Un'efficiente implementazione di timer assoluto . . . . .	63
4.4.1	Timer: definizione ed uso nei microcontrollori . . . . .	64
4.4.2	Il Real Time Clock . . . . .	65
4.4.3	Il Virtual High resolution Time . . . . .	65
4.5	Temporizzazione del bsp . . . . .	68
4.6	Cenni implementativi del controllore . . . . .	69
<b>5</b>	<b>Risultati sperimentali</b>	<b>73</b>
5.1	Quantificazione del jitter indotto dall'hardware . . . . .	73
5.2	Valutazione delle performance di FLOPSYNC . . . . .	76
5.2.1	Boot del nodo . . . . .	76
5.2.2	Distribuzione statistica dell'errore di sincronizzazione . . . . .	77
5.3	Comportamento sotto variazioni di temperatura . . . . .	79
5.3.1	Comparazione tra FLOPSYNC, FTSP e FBS . . . . .	79
5.3.2	Efficacia del tool di configurazione FLOPSYNC . . . . .	81
<b>6</b>	<b>Modello del consumo di potenza</b>	<b>83</b>
<b>7</b>	<b>Conclusioni e punti aperti</b>	<b>89</b>





# Elenco delle figure

2.1.1 Orientamento degli assi in un cristallo di quarzo. . . . .	8
2.1.2 Tipologie di tagli di un quarzo. . . . .	9
2.1.3 Modalità di vibrazione . . . . .	10
2.1.4 Circuito elettrico equivalente per un oscillatore al quarzo. . . . .	10
2.1.5 Variazioni di frequenza di lungo periodo . . . . .	12
2.1.6 Variazione di frequenza di breve periodo . . . . .	12
2.1.7 Caratteristica frequenza-temperatura al variare dei tagli. . . . .	13
2.1.8 Variazioni di breve periodo legate a condizioni ambientali. . . . .	14
2.3.1 Tassonomia degli schemi di sincronizzazione in sintesi. . . . .	19
2.4.1 Dipendenze lineari e vincoli imposti ad $a_{12}$ e $b_{12}$ con tre data point. . . . .	24
2.6.1 Errore di sincronizzazione in assenza di meccanismo di compensazione. . . . .	30
3.2.1 Sistema da controllare nel problema base. . . . .	33
3.2.2 Schema di controllo per il nodo $i$ -esimo. . . . .	37
3.2.3 Azione della componente termica del disturbo sul controllo. $TM$ indica il modello termico di equazioni 3.2.27, mentre il blocco $QUAR-ZO$ rappresenta la caratteristica temperatura-frequenza del quarzo. . . . .	41
3.2.6 Coppie $(T, \alpha)$ per il caso 1 che rendono il progetto attuabile. . . . .	44
3.2.7 Coppie $(T, \alpha)$ per il caso 2 che rendono attuabile il progetto. . . . .	44
3.2.4 Caso 1: errore di picco (sopra) e tempo di recovery (sotto) come funzione di $T$ ed $\alpha$ . . . . .	45
3.2.5 Caso 2: errore di picco (sopra) e tempo di recovery (sotto) come funzione di $T$ ed $\alpha$ . . . . .	46
3.3.1 Struttura del controllo decentralizzato; $j_i^f(k)$ rappresenta il jitter che ha origine dallo schema di flooding, mentre $XTAL$ e $SW$ rappresentano quelli introdotti dal quarzo e dal software. . . . .	47
3.3.2 Traslazione tra tempo locale e tempo globale con clock virtuale monotono. . . . .	50
3.4.1 Collocazione di FLOPSYNC all'interno dello stack radio ZigBee. . . . .	52
3.4.2 Modello del sistema controllato con finestra. . . . .	53

4.1.1 Tool di configurazione per il controllore. . . . .	58
4.1.2 Output del tool di configurazione riferito alla figura 4.1.1 per la scelta $T, \alpha$ . . . . .	59
4.2.1 Architettura di un mote. . . . .	59
4.2.2 Alcuni mote usati per gli esperimenti. . . . .	61
4.3.1 Vista schematizzata del formato del frame IEEE 802.15.4 [1] . . . .	62
4.4.1 Schema a blocchi di un comune timer da microcontrollore. . . . .	64
4.4.2 Modello del timer. . . . .	67
5.1.1 Distribuzioni empiriche cumulative tra i tempi di interrupt del trans- ceiver CC2520. . . . .	75
5.2.1 Errore di sincronizzazione durante la fase di boot di FLOPSYNC, con il controllore deadbet (in nero) e senza in (rosso). . . . .	76
5.2.2 Media e varianza per l'errore un istante prima ed un istante dopo la sincronizzazione, in funzione degli hop. . . . .	78
5.3.1 Comparazione tra FLOPSYNC (in nero), FTSP (rosso) e FBS (blu) in una transizione ombra-sole. I punti evidenziano l'errore ad $e_i(t(k+1)^-)$ , mentre le linee mostrano l'errore di sincronizzazione all'interno di un periodo di sincronizzazione. La linea orizzontale tratteggiata delimita l'area dove l'errore è all'interno di $20\mu s$ . I nodi sono esposti al sole al minuto 10. . . . .	80
5.3.2 Errore di sincronizzazione durante una variazione di temperatura. . .	81
6.0.1 Stati operativi della CPU e della radio. . . . .	84
6.0.2 Consumo di corrente tracciato per il nodo reference (non in scala), radio (in alto) e CPU (in basso). . . . .	85
6.0.3 Consumo di corrente tracciato per il nodo non-reference (non in scala), radio (in alto) e CPU (in basso). . . . .	86

# Elenco delle tabelle

2.1	Classificazione degli schemi di sincronizzazione . . . . .	26
2.2	Comparazione delle performance degli schemi di sincronizzazione. Per la complessità, $n$ indica il numero di nodi, $m$ il numero di sincronizzazioni, $C$ il numero di <i>cluster head</i> [2], $v$ il numero di vicini dati dall'algoritmo spanning-tree e $k$ è una costante che indica il numero di campioni memorizzati per la regressione lineare. . . . .	29
3.1	Specifiche di progetto per il caso 1 ed il caso 2. . . . .	44
4.1	Comparazione STM32vdiscovery-Telosb . . . . .	60
5.1	Media e deviazione standard dell'errore e della finestra di sincronizzazione per un esperimento multi-hop con periodo di 60s e durata di 6 giorni. . . . .	77





# Capitolo 1

## Introduzione

Il mondo in cui si vive è pieno di sensori. Gli edifici in cui si lavora sono dotati di sensori che monitorano la temperatura, rilevano la presenza di persone, fumo e fuoco e controllano la sicurezza. Le auto contengono dozzine, se non centinaia, di sensori che controllano le prestazioni del motore, i freni, i dispositivi di sicurezza per i passeggeri, solo per citarne alcuni. Negli ultimi decenni i sensori sono diventati molto più piccoli, costano e consumano meno, in parte anche grazie alla legge di Moore e alla rivoluzione dei MEMS<sup>1</sup>. Il costo per la posa dei cavi necessari per l'alimentazione e la trasmissione dei dati è quasi sempre di gran lunga superiore a quello del sensore stesso e non sempre è praticabile.

A partire dagli anni '90, grazie ai notevoli passi in avanti fatti nel campo dei sistemi embedded e alla conseguente diminuzione dei costi dei dispositivi elettronici, è nato e progressivamente cresciuto l'interesse per le reti di sensori wireless (dall'inglese Wireless Sensor Networks WSN). In questi ultimi anni le WSN hanno aperto scenari di utilizzo decisamente innovativi che spaziano dal monitoraggio ambientale alla salute della persona, dalla domotica all'agricoltura e chissà quali e quanti altri utilizzi ci potranno essere in un futuro che si prospetta sempre più "interconnesso". La crescita spasmodica e l'utilizzo piuttosto trasversale e dissimile, rende alquanto appetitoso per la comunità scientifica lo studio delle problematiche inerenti le reti di sensori. Uno tra questi riguarda la sincronizzazione degli orologi in ambito distribuito.

---

<sup>1</sup>La sigla MEMS sta per Micro Electro-Mechanical Systems ed indica quello che la tecnologia del microscopico ha prodotto (si intende quindi che la dimensione media degli oggetti considerati sia di un micrometro), consentendo di rendere la nanotecnologia una realtà.

## 1.1 Definizione di una WSN

Una WSN è una rete formata da dispositivi piuttosto "semplici", perfettamente autonomi che cooperano nell'esecuzione di una qualche applicazione di raccolta informazioni di un determinato fenomeno fisico. Il termine sensore sta ad indicare un dispositivo che è in grado di tradurre una grandezza fisica in un segnale di natura diversa (tipicamente elettrico) più facilmente misurabile o memorizzabile. Solitamente il segnale originale viene tradotto in una grandezza di tensione o corrente che ne permette l'acquisizione e l'eventuale elaborazione in formato digitale.

Le reti di sensori sono costituite da un insieme di nodi. Ciascun nodo della rete è dotato di uno o più sensori, di un trasmettitore radio (o altro tipo di dispositivo di trasmissione wireless), di un piccolo microcontrollore (che permette l'elaborazione delle informazioni) e infine di una sorgente di energia (tipicamente una batteria). I nodi di una WSN sono spesso impropriamente chiamati sensori, ciò ne evidenzia il fine principale: la rilevazione delle condizioni ambientali dello spazio in cui operano.

Lo sviluppo delle WSN fu originariamente motivato dalle possibili applicazioni in campo militare, come ad esempio la sorveglianza di un campo di battaglia. Tuttavia le reti di sensori hanno trovato ampio spazio in molte applicazioni civili, come monitoraggio di ambienti e abitazioni, applicazioni healthcare, automazione e controllo del traffico. La vasta varietà di sistemi ed applicazioni in cui trovano impiego le WSN, rende difficoltoso discutere su applicazioni specifiche e direzione della ricerca in questo ambito [3].

Le WSN sono caratterizzate da alcuni aspetti chiave come, per esempio, i vincoli di potenza elaborativa, la durata limitata della batteria, un basso duty-cycle e connessioni multi-a-uno che creano delle sfide progettuali nell'ambito delle telecomunicazioni.

Il costo di una rete di sensori è variabile. Dipende dal numero di nodi, dalle dimensioni di questi (generalmente più piccoli sono più costano) e dal tipo di sensori montati su ciascun nodo. Altri vincoli legati alla dimensione ed al costo del nodo corrispondono alle quantità di risorse che il nodo dispone, come ad esempio l'energia, la memoria, capacità computazionale e la banda di trasmissione.

## 1.2 Scenari applicativi

Sebbene storicamente il principale utilizzo delle reti di sensori wireless sia stato in campo militare riconducibile alla guerra fredda, oggi è possibile individuare una più vasta area applicativa per questa tecnologia. La principale suddivisione, in termini applicativi delle WSN, dipende dalla presenza di una qualche forma

di retroazione o controllo sull'ambiente stesso, permettendo quindi di distinguere tra applicazioni di *monitoraggio* ed applicazioni di *controllo*. Ogni volta che una WSN si occupa della sola raccolta dati, parleremo di applicazione di monitoraggio. Si parlerà invece di controllo quando la rete stessa avrà il compito di interagire in maniera attiva con il fenomeno fisico che stiamo analizzando. Gli utilizzi più diffusi di applicazioni WSN sono:

- Ambito militare
  - monitoraggio forze nemiche
  - monitoraggio forze alleate ed equipaggiamenti
  - sorveglianza di campi di battaglia
  - tracking di obiettivi
  - stima dei danni
  - rilevazione di attacchi nucleari e biochimici
  
- Ambito ambientale
  - rilevamento di catastrofi naturali
  - monitoraggio di microclimi
  - agricoltura di precisione
  
- Ambito biomedico:
  - monitoraggio a distanza di parametri fisiologici
  - tracking di medici e pazienti all'interno di un ospedale
  - gestione dei medicinali
  
- Ambito domestico:
  - Home Automation
  - lettura contatori (Smart Metering)
  - sorveglianza
  
- Ambito industriale:
  - controllo ambientale in costruzioni industriali ed uffici
  - controllo dell'inventario

- tracking di veicoli
- monitoraggio della supplychain
- Ambito dei trasporti:
  - monitoraggio del traffico
  - sensori intraveicolo
- Ambito automotive
  - controllo del veicolo
  - infotainment

### 1.3 La sincronizzazione nel dominio WSN

La sincronizzazione temporale è un elemento critico ed importante per ogni sistema distribuito. Il concetto di tempo deriva da quello più generale che riguarda l'ordine in cui avvengono gli eventi. Per questo, gran parte dei dati raccolti da una rete di sensori non porta alcuna informazione se non accompagnata da un riferimento temporale omogeneo. Molte applicazioni per reti di sensori richiedono che l'orologio locale di ciascun nodo sensore sia sincronizzato secondo vari gradi di precisione.

Le proprietà intrinseche di una rete di sensori come risorse di energia, calcolo, memoria, banda limitata combinate con le alte densità dei nodi della rete rendono i metodi tradizionali di sincronizzazione inadeguati. Gli esistenti metodi di sincronizzazioni necessitano di essere estesi per soddisfare le nuove richieste. Tutto ciò porta ad una continua ricerca di algoritmi specifici di sincronizzazione per reti di sensori.

Da una temporizzazione accurata dipendono applicazioni critiche o servizi della rete stessa, un esempio su tutti il TDMA<sup>2</sup> schedules. Quando l'accesso al mezzo trasmissivo, piuttosto che il meccanismo di power saving sono regolati con la tecnica del TDMA, disporre di un algoritmo di sincronizzazione efficiente diventa il punto cardine per l'ottimizzazione delle risorse del nodo.

Altre applicazioni che richiedono un'elevata accuratezza di sincronizzazione sono quelle di geo-localizzazione. Sebbene il GPS (Global Positioning System) sia un accurato sistema di stima della posizione, purtroppo non si può utilizzare in ambienti indoor poiché usa segnali provenienti da satelliti GPS. Diventa chiaro

---

<sup>2</sup>Time Division Multiple Access è una tecnica di moltiplicazione numerica in cui la condivisione del canale è realizzata mediante ripartizione del tempo di accesso allo stesso.

quindi come una localizzazione accurata e a basso costo sia un requisito critico per un corretto funzionamento di una rete di sensori wireless in ambienti chiusi. Alcune tecniche di localizzazione *time based* stimano la distanza tra due stazioni misurando il tempo di propagazione del segnale. Per avere un sistema in grado di assicurare una precisione dell'ordine del centimetro occorre essere in grado di rilevare intervalli di tempo dell'ordine delle decine di picosecondi. Oltretutto, è evidente la necessità di una perfetta sincronizzazione tra i dispositivi della rete.

## 1.4 Obiettivi e contenuto

Nel corso degli anni diverse soluzioni sono state proposte dalla comunità scientifica internazionale al problema della sincronizzazione con i loro pregi e difetti. In questo lavoro viene presentato uno schema innovativo di sincronizzazione per WSN basato su una prospettiva di progettazione avveniristica, denominato FLOPSYNC acronimo di Feedback LOw Power SYNChronization. Si tratta di uno schema significativamente semplice rispetto a quelli ad oggi esistenti, garantisce alta precisione ed un consumo di potenza estremamente ridotto. Con FLOPSYNC viene anche introdotto il concetto di monotonicità temporale, fino ad oggi problematica non solo irrisolta ma addirittura mai menzionata prima d'ora.

Le fasi seguite, al fine di perseguire gli obiettivi sintetizzati col termine FLOPSYNC, sono state riprese nell'ordine in cui è stato strutturato l'elaborato di tesi. Pertanto nel capitolo 2 viene proposto al lettore dapprima una rassegna nozionistica sul quarzo, utile a comprendere i fattori che causano l'assenza di sincronia in una WSN, poi viene definita la notazione adottata per il resto del lavoro e infine vengono presentati i lavori più significativi in questo contesto e le motivazioni che hanno portato alla nascita di FLOPSYNC.

Il capitolo 3 è la parte centrale di questa trattazione e ripercorre quelle fasi che potremmo definire studio del problema, modellazione e progettazione.

Nel capitolo 4 dopo una breve presentazione dell'architettura hardware/software su cui FLOPSYNC è stato implementato, si proporranno al lettore brevi frammenti di interessanti soluzioni implementative.

Il capitolo 5 è il frutto dei risultati sperimentali ottenuti dalla valutazione di FLOPSYNC. Verranno ripresi alcuni dei protocolli presentati nello stato dell'arte per la comparazione, a dimostrazione delle migliori introdotte.

Il capitolo 6 racchiude in se uno dei risultati più importanti raggiunti con questo lavoro: si tratta del consumo di potenza. FLOPSYNC aggiunge un overhead di consumo da essere ritenuto irrisorio.

Infine il capitolo 7 commenta brevemente tutto il lavoro e propone alcuni interessanti spunti per il futuro.

# Capitolo 2

## Stato dell'arte

Questo capitolo pone le fondamenta per la comprensione dell'intero lavoro. Verranno illustrate le caratteristiche primarie del quarzo ed i protocolli più significativi ad oggi esistenti. Il lettore in possesso di queste conoscenze è comunque invitato alla lettura delle sezioni seguenti in quanto viene definita la terminologia usata nel resto del presente lavoro. Inoltre viene accennato un confronto tra FLOPSYNC e gli altri protocolli i cui aspetti risulteranno chiari col prosieguo della lettura dei capitoli successivi.

### 2.1 Oscillatore al quarzo

Un oscillatore al quarzo è un circuito elettronico che usa la risonanza meccanica di un cristallo piezoelettrico vibrante per ottenere un segnale elettrico caratterizzato da una frequenza molto precisa. Tipicamente il cristallo utilizzato è il quarzo.

#### 2.1.1 Terminologia tecnica

##### **Piezoelettricità**

La proprietà principale di un cristallo che lo rende usabile come un risonatore prende il nome di piezoelettricità. Il funzionamento di un cristallo piezoelettrico è abbastanza semplice: quando viene applicata una pressione (o decompressione) esterna, si posizionano, sulle facce opposte, cariche di segno opposto. Quindi il cristallo si comporta come un condensatore al quale è stata applicata una differenza di potenziale. Se le due facce vengono collegate ad un circuito esterno, viene generata una corrente elettrica detta corrente piezoelettrica. Al contrario, quando si applica una differenza di potenziale al cristallo, esso si espande o si contrae.

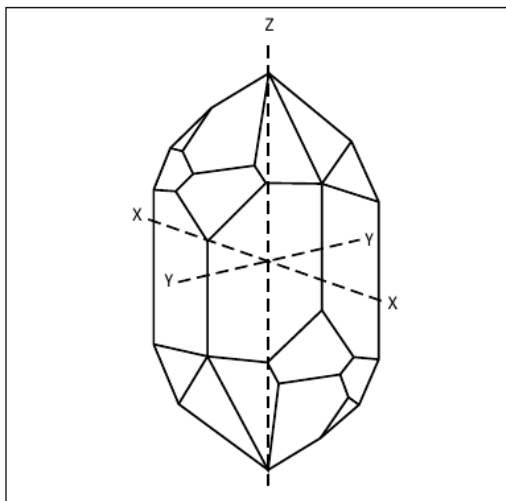


Figura 2.1.1: Orientamento degli assi in un cristallo di quarzo.

### Struttura del quarzo

Il cristallo di quarzo è formato da silicio e ossigeno ( $\text{SiO}_2$ ). La sua forma caratteristica è il risultato delle celle unitarie con cui il cristallo cresce. Queste celle unitarie sono identiche e sono costituite da atomi disposti in un disegno geometrico ripetitivo. I cristalli di quarzo hanno un corpo geometrico tridimensionale. La maggior parte delle proprietà fisiche di un cristallo sono anisotropiche (dipendenti dalla direzione), perciò vi saranno imperfezioni nelle fasi di compressione e decompressione del cristallo legate all'anisotropia. Una variazione del coefficiente piezoelettrico, per esempio, crea un confine attraverso il quale il segno della carica differisce quando viene applicata tensione. Questo fenomeno meglio conosciuto come *twin boundary*, impedisce al pezzo di cristallo di risonare e lo rende inadatto per un oscillatore. Poiché una notevole quantità di lavoro è spesa per fare un buon risonatore al quarzo, questi difetti devono essere rilevati in fase di costruzione. L'orientamento cristallino piuttosto che la presenza di difetti come il *twin boundary* o fratture vengono rilevati attraverso l'uso di luce polarizzata, raggi X e attacco chimico.

L'asse di maggior crescita del quarzo è chiamato asse ottico. Questo asse non è anisotropo alla luce, quindi la luce passa facilmente. Ai fini del taglio di pezzi di quarzo per agire come risonatori, l'asse ottico è etichettato come l'asse Z in un sistema ortogonale di coordinate X, Y, Z. Un cristallo di quarzo con 6 lati ha tre assi X distinti, tre assi Y definiti attorno all'asse Z distanziati tra loro di circa  $120^\circ$ . Gli assi Y sono perpendicolari alle facce del prisma, mentre gli assi X intersecano gli angoli adiacenti alle facce del prisma (figura 2.1.1).



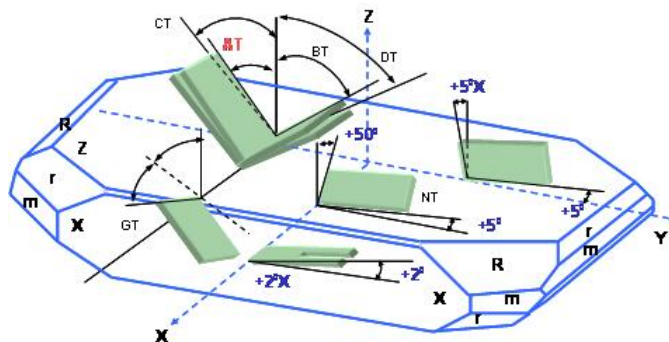


Figura 2.1.2: Tipologie di tagli di un quarzo.

### Tagli del quarzo

Un opportuno frammento di quarzo viene ottenuto tagliando il cristallo secondo assi ed angoli specifici. La scelta di asse ed angoli determina i parametri fisici ed elettrici per il risonatore. Numerosi tagli possono essere fatti semplicemente cambiando l'angolo e l'asse di riferimento, come mostra la figura 2.1.2.

### Modalità di vibrazione

Quando un cristallo è sottoposto a tensione questo inizia a vibrare. La modalità di vibrazione dipende dal modo in cui il cristallo è stato tagliato e dallo spessore. Varie modalità di vibrazione sono riportate in figura 2.1.3.

### Montaggio del quarzo

La struttura portante ed i modi utilizzati per ottenere i contatti elettrici sono dettati dalla modalità di vibrazione. Al fine di evitare l'errata vibrazione del cristallo, una cura scrupolosa deve essere esercitata nel montaggio; il sostegno non deve diventare una parte del risonatore poiché potrebbe assorbire energia.

### Circuito equivalente

Il circuito equivalente di figura 2.1.4 fornisce il collegamento tra le proprietà fisiche di un quarzo e l'area di applicazione, l'oscillatore. Le costanti fisiche del quarzo determinano il valore di  $C_0$ ,  $C_1$ ,  $L_1$ ,  $R_1$ . Un oscillatore al quarzo ha l'ulteriore vantaggio che le sue costanti elastiche e la sua dimensione variano in maniera molto meno sensibile rispetto ad un circuito elettronico.

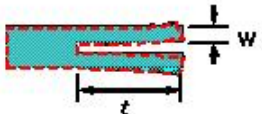
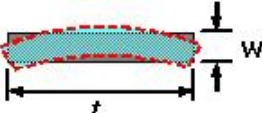
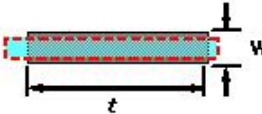
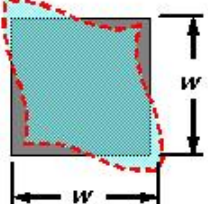
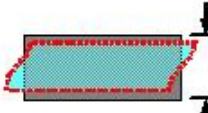
Modalità di vibrazione	Orientamento angolo
Tuning Fork 	$+ 2^\circ X$
Flexure 	XY NT
Extension 	$+ 5^\circ X$ $- 18.5^\circ X$
Face Shear 	DT CT SL
Thickness Shear 	AT Fundamental AT 3 <sup>rd</sup> Overtone AT 5 <sup>th</sup> Overtone BT Fundamental

Figura 2.1.3: Modalità di vibrazione

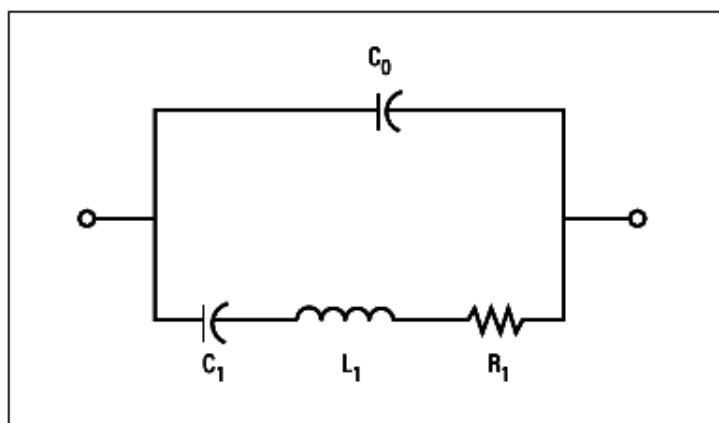


Figura 2.1.4: Circuito elettrico equivalente per un oscillatore al quarzo.

### 2.1.2 Frequenza di oscillazione

Il risultato desiderato dal quarzo e dal suo circuito oscillatore associato è la generazione di una frequenza precisa. La frequenza del quarzo, tuttavia, è determinata dallo spessore, densità, elasticità, cambiamenti molecolari, e dalla zona di risonanza su cui la piastra di quarzo è in funzione. Poiché questi fattori sono influenzati da variazioni di temperatura, tempo, energia ed altre condizioni ambientali, è ragionevole aspettarsi che la frequenza di oscillazione del quarzo vari nel tempo.

Un modo utile per caratterizzare la stabilità di un oscillatore è esprimendo la variazione di frequenza rispetto al tempo. La stabilità è generalmente espressa come la quantità di frequenza che cambia nel corso di un periodo di tempo (espressa in parti per milione) e si è soliti distinguere variazioni di lungo periodo e variazioni di breve periodo.

#### Variazioni di lungo periodo

La graduale variazione di frequenza per giorni o mesi è nota come *invecchiamento*. Questo si verifica per vari motivi, ad esempio, le proprietà fisiche del materiale su cui è montato il cristallo possono cambiare. Il coefficiente di elasticità del cristallo si modifica quando è sottoposto a stress, o quando vi è fuga dei gas intrappolati, o quando contaminazioni attaccano o lasciano la superficie del cristallo. L'invecchiamento si verifica ad un tasso relativamente costante per decade per ogni cristallo (figura 2.1.5). Pertanto, per mantenere una frequenza precisa, periodiche regolazioni devono essere effettuate. Generalmente, la frequenza di un oscillatore può essere variata di qualche ciclo per mezzo di un condensatore regolabile posto in retroazione. Un oscillatore a  $10\text{MHz}$  con un campo di regolazione di  $20\text{Hz}$  può essere corretto per 75.000 ore (circa 9 o 10 anni) di invecchiamento al tasso di  $5 \times 10^{-10}\text{Hz}$  al giorno.

#### Variazioni di breve periodo

Le variazioni di frequenza di breve periodo sono una misura della frequenza o rumore di fase. Questa viene specificata come deviazione standard delle fluttuazioni di frequenza per un tempo medio specifico. Queste piccole variazioni di frequenza sono essenzialmente sovrapposte sulla curva di invecchiamento (figura 2.1.6).

Il maggior responsabile delle variazioni di breve periodo è il range di temperatura al quale il quarzo opera. Un oscillatore posto esattamente alla temperatura di  $25^\circ\text{C}$  ed una variazione di frequenza di 5 parti per milione (ppm) per grado Celsius, potrebbe subire uno spostamento di frequenza di 25ppm con l'aumento di soli  $5^\circ\text{C}$  di temperatura. La variazione di frequenza è dovuta al coefficiente di

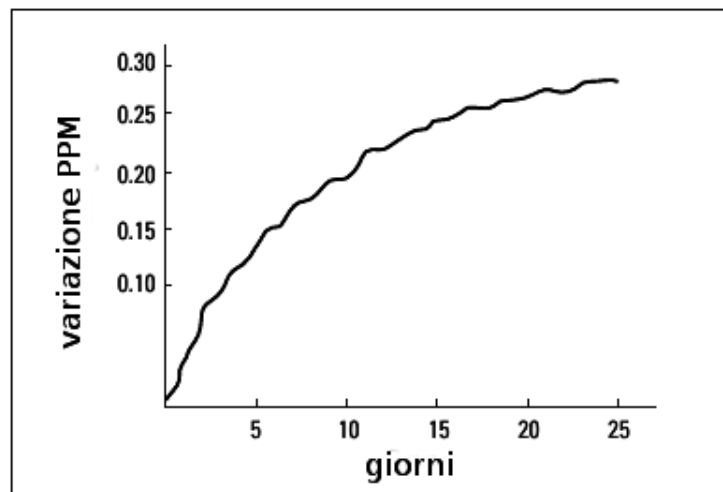


Figura 2.1.5: Variazioni di frequenza di lungo periodo

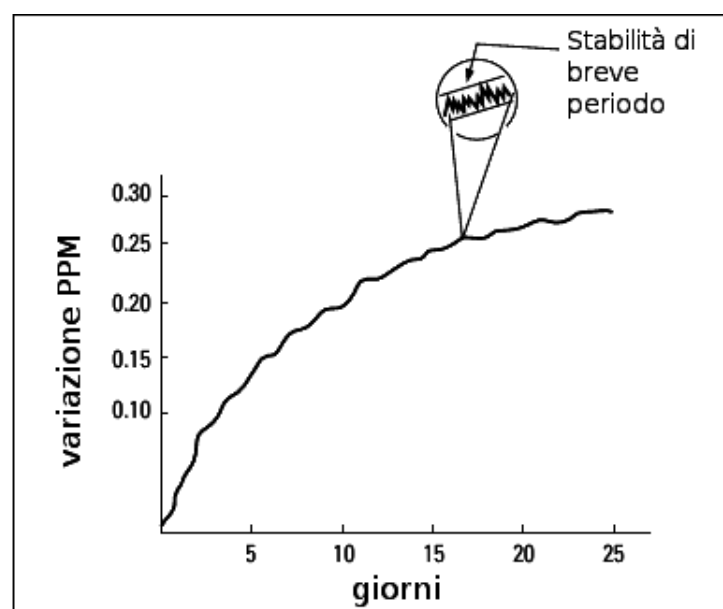


Figura 2.1.6: Variazione di frequenza di breve periodo

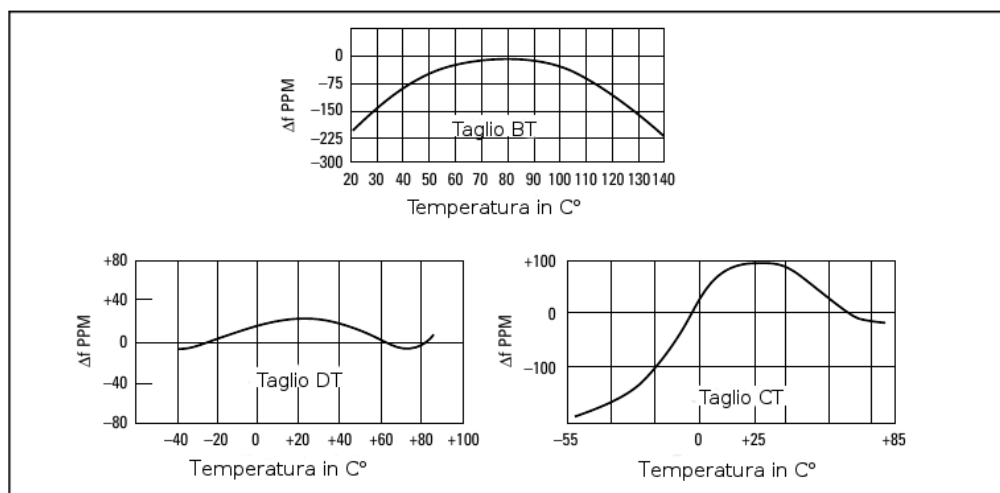


Figura 2.1.7: Caratteristica frequenza-temperatura al variare dei tagli.

temperatura del quarzo e quindi dipende dal taglio del quarzo. Grafici di temperatura rapportati alla variazione di frequenza per vari tagli, sono riportati nella figura 2.1.7. Ovviamente, non è possibile evitare completamente queste variazioni di frequenza se il cristallo deve essere utilizzato in un ampio intervallo di temperatura. Pertanto altre tecniche devono essere utilizzate per ridurre questo effetto, ad esempio tecniche di compensazione.

Oltre che dalla temperatura le variazioni di breve periodo sono influenzate, se pur in misura nettamente inferiore, da:

- variazioni di energia applicata al quarzo,
- orientamento del quarzo sottoposto alla forza gravitazionale,
- interferenze elettromagnetiche,
- esposizione del quarzo a vibrazioni meccaniche,
- shock che provocano uno stress improvviso sul quarzo e ne deformano temporaneamente la struttura di montaggio,
- on/off dell'oscillatore genera un ritmo non costante di invecchiamento.

La figura 2.1.8 riassume i fenomeni ambientali che provocano variazioni di breve periodo.

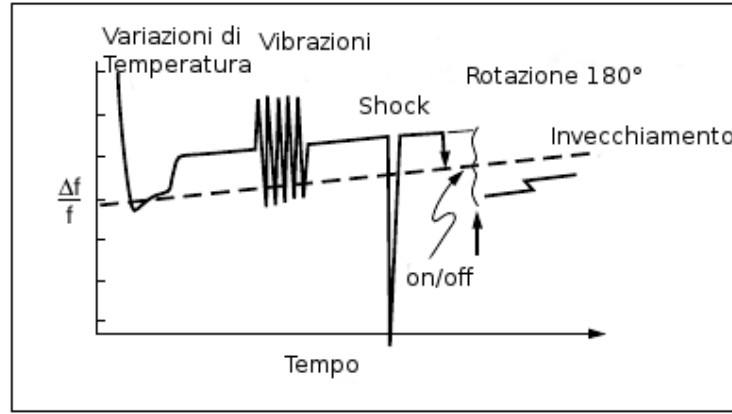


Figura 2.1.8: Variazioni di breve periodo legate a condizioni ambientali.

## 2.2 Notazione e terminologia

In questa sezione verranno definite le principali notazioni che d'ora in avanti saranno utilizzate per definire e studiare gli aspetti legati alla sincronizzazione.

Denotiamo con  $t$  il tempo universale espresso in secondi. Si definisce *offset* la differenza di tempo espressa in secondi esistente tra due clock ad un dato istante ( $C_A(t) - C_B(t)$ ) [4]. Per *clock* si intende un orologio atto a contare il tempo, più precisamente un dispositivo elettronico che conta le oscillazioni di un quarzo che lavora ad una frequenza ben precisa. Il clock di un apparecchio elettronico fondamentale è un timer che conta le oscillazioni del quarzo per mezzo di un registro contatore.

Si definisce *skew* la differenza di frequenza tra due clock, ossia la derivata prima rispetto al tempo dell'offset ( $\dot{C}_A(t) - \dot{C}_B(t)$ ). Clock reali mostrano anche variazioni dello skew rispetto al tempo che pertanto chiameremo *drift*, ossia la derivata seconda rispetto al tempo dell'offset ( $\ddot{C}_A(t) - \ddot{C}_B(t)$ ) [5]. In aggiunta, in questo lavoro occorrerà tener conto delle variazioni di frequenza di breve periodo che chiameremo *jitter*.

Chiameremo *nodo reference* il nodo il cui clock sarà da intendersi esatto e costante che definisce il tempo  $t$  *universale*, a cui tutti gli altri nodi  $i$ -esimi dovranno sincronizzarsi. Sia poi  $f_o$  la frequenza di targa del clock del nodo reference tipicamente espressa nel datasheet del suo oscillatore al quarzo misurata in Hertz, ed  $f_i(t)$  quella relativa al nodo  $i$ -esimo al tempo  $t$ . Infine indichiamo con  $t_i(t)$  il *tempo locale* del nodo  $i$ -esimo ed  $o_i(t)$  l'offset del nodo  $i$ -esimo, entrambi riferiti all'istante  $t$ . Per comodità useremo un indice  $k$  da intendersi come multiplo intero di un intervallo di tempo  $T$ , dove  $T$  rappresenta il periodo di tempo che trascorre tra una sincronizzazione e la precedente. Raggruppiamo ora  $s_i(t)$  e  $j_i(t)$  in un singolo

disturbo in frequenza in questo modo:  $\delta_f(t) = s_i(t) + j_i(t)$  cosicché da esprimere l'errore tra clock locale e clock globale al nodo  $i$  -esimo come  $e_i(t) = t_i(t) - t$  per cui abbiamo

$$e_i(t) = o_i + \int_0^t \frac{\delta_f(\tau)}{f_o} d\tau \quad (2.2.1)$$

Assumiamo che ad un certo istante  $t(k)$  del tempo universale venga eseguito un algoritmo di sincronizzazione per ridurre istantaneamente l'errore  $e_i(t(k)^+)$ , dove il simbolo "+" indica l'errore calcolato un istante dopo la sincronizzazione. Utilizzando la medesima notazione, possiamo calcolare l'errore un istante prima della sincronizzazione come:

$$e_i(t(k+1)^-) = e_i(t(k)^+) + \int_{t(k)}^{t(k+1)} \frac{\delta_f(\tau)}{f_o} d\tau, \quad (2.2.2)$$

e tale meccanismo che riduce l'errore al tempo  $t(k+1)^-$  sarà chiamato *compensazione skew/drift*.

Tradizionalmente, la sincronizzazione di clock è realizzata trasmettendo un timestamp dal nodo reference verso tutti gli altri. Tramite un opportuno *schema di flooding*<sup>1</sup>, si garantisce che i nodi non-reference ricevono il timestamp. Una volta ricevuto il timestamp, i nodi lo impiegano per ricalcolare il loro tempo locale sia con sovrascrittura diretta del clock che senza.

Ai fini della sincronizzazione è di fondamentale importanza considerare anche il *ritardo di trasmissione*, che possiamo definire come il tempo che intercorre tra la generazione del timestamp nel nodo reference e la ricezione del timestamp nel nodo non-reference. Il ritardo di trasmissione si può esprimere come la somma di 5 contributi differenti del segmento di comunicazione:

1. ELABORAZIONE DEL SEGNALE D'INVIO, è l'intervallo di tempo impiegato per percorrere lo stack radio dal livello più alto di costruzione del messaggio fino a quello di richiesta del canale;
2. ACCESSO AL MEZZO TRASMISSIVO, si riferisce al tempo necessario prima che effettivamente il messaggio venga trasmesso. Tale tempo solitamente è dovuto alla contesa del mezzo trasmissivo oppure alla coda di messaggi in attesa di trasmissione;

---

<sup>1</sup>Il flooding è una tecnica comune usata per la ricerca di cammini e per la disseminazione delle informazioni nelle reti ad hoc wireless e wired. Il flooding usa un approccio reattivo dove ogni nodo che riceve un pacchetto di controllo lo rinvia a tutti i suoi vicini.

3. TRASMISSIONE DATI, è il tempo necessario per trasmettere un pacchetto e può essere facilmente ottenuto dalla conoscenza della lunghezza del messaggio da trasmettere e dal data rate di trasmissione;
4. PROPAGAZIONE RADIO, indica il tempo di volo di un messaggio tra due nodi;
5. ELABORAZIONE DEL SEGNALE DI RICEZIONE, coincide con il tempo di risalita dello stack architetturale del messaggio, dall'arrivo nel livello fisico al livello più alto.

Indipendentemente dall'efficienza dello schema di flooding utilizzato, piuttosto che dalla precisione con cui viene generato il timestamp [6], il ritardo di trasmissione può essere solo stimato con un certo grado d'incertezza, sia che si usi una rete cablata per la trasmissione sia che si usi una wireless.

Definiamo *monotonicità del clock* la proprietà di uno schema di sincronizzazione atta a mantenere l'ordinamento tra tempo locale e tempo globale.

Nel resto di questa trattazione l'uso dei termini *schema* e *protocollo* potrebbero apparire agli occhi del lettore come sinonimi, in realtà si vuole indicare con il primo termine solo gli elementi essenziali di un modello di sincronizzazione mentre col secondo l'insieme di regole che governano l'intera sincronizzazione.

## 2.3 Tassonomia degli schemi di sincronizzazione

Gli schemi di sincronizzazione per WSN fino ad oggi proposti, possono essere classificati in base ai seguenti criteri [7] :

### Modello di comunicazione

- *master-slave (M-S)*: un nodo viene definito come master e gli altri come slave. Il clock del nodo master, chiamato anche nodo reference, viene assunto come globale.
- *peer-to-peer (P-P)*: ogni nodo può comunicare direttamente con gli altri nodi presenti nella rete eliminando in questo modo i rischi legati alla caduta del nodo master. Sicuramente più flessibile ma molto più difficile da controllare.

### Tipologia di sincronizzazione

- *interna (I)*: ha come obiettivo quello di minimizzare la differenza massima tra il clock dei vari nodi. Può essere fatta sia in master-slave che peer-to-peer.



- *esterna (E)*: esiste una sorgente esterna che fornisce un clock estremamente preciso del mondo reale di riferimento a cui tutti gli altri nodi dovranno sincronizzarsi. Esclude pertanto un modello di comunicazione peer-to-peer.

### Modello di sincronizzazione

- *sender to receiver (S-R)*: il nodo sender manda periodicamente un messaggio al nodo receiver; quest'ultimo si sincronizza con il nodo sender usando il messaggio ricevuto. Lo svantaggio di questo approccio è l'inconsistenza dei ritardi di trasmissione tra sender e receiver.
- *receiver to receiver (R-R)*: si sfruttano le proprietà broadcast del mezzo fisico assumendo che, se due nodi ricevono lo stesso messaggio in una trasmissione, lo ricevono approssimativamente allo stesso istante. In questo modo il sender invia un messaggio a due receiver e la sincronizzazione viene effettuata considerando la differenza tra i clock locali dei due receiver. Il vantaggio di questo approccio è quello di ridurre le inconsistenze dovute ai ritardi di trasmissione.

### Topologia di rete

- *single-hop (S-H)*: in una rete single-hop, un nodo sensore può comunicare direttamente e scambiare messaggi con qualsiasi altro sensore nella rete. La rete è spesso troppo grande, rendendo impossibile per ogni nodo sensore scambiare direttamente messaggi con ogni altro nodo.
- *multi-hop (M-H)*: la rete risulta composta fisicamente da più hop, pertanto i nodi possono comunicare solo con i loro "vicini".

### Dinamicità della rete

- *rete statica (RS)*: una volta dislocati i nodi la topologia della rete nell'ambiente non cambia più.
- *rete dinamica (RD)*: in una rete dinamica i nodi hanno la capacità di "muoversi", e di comunicare con altri sensori solo quando entrano nel range di comunicazione di quest'ultimi. Alcuni schemi richiedono il ricalcolo topologico della rete periodico.

### Ottimizzazione

- *piggybacking* ( $P$ ): offre la possibilità di incapsulare altre informazioni nel pacchetto di sincronizzazione utili ad esempio per fornire informazioni e/o comandi per l'intera rete.
- *no piggybacking* ( $\neg P$ ): non permette di inviare null'altro che il pacchetto di sincronizzazione.

### Misurazione dell'offset

- *offset probabilistico* ( $OP$ ): il valore dell'offset viene calcolato statisticamente. Un approccio di questo tipo viene utilizzato perché permette al protocollo di sincronizzazione di inviare pochi messaggi e di conseguenza basso utilizzo delle risorse e basso consumo.
- *offset deterministico* ( $OD$ ): garantisce un upper bound di clock offset per mezzo di algoritmi deterministici.

### Tipologia di correzione del clock

- *senza compensazione del clock* ( $C\nrightarrow C$ ): il clock viene corretto solo l'istante dopo la sincronizzazione, per cui l'errore  $e(t(k)^+$  (ossia un'istante dopo la sincronizzazione) sarà minimo da questo punto in poi tenderà ad aumentare raggiungendo il suo massimo all'istante  $t(k+1)^-$ .
- *con compensazione del clock* ( $CC$ ): viene fatta compensazione del clock su tutto il periodo di sincronizzazione con lo scopo di minimizzare l'errore qualsiasi istante di tempo si consideri.

### Modalità di aggiornamento del clock

- *clock legato* ( $CL$ ): il clock locale viene aggiornato a quello globale dopo aver eseguito il processo di sincronizzazione, per mezzo di una sovrascrittura del clock.
- *clock slegato* ( $CS$ ): le informazioni per convertire il clock locale in quello globale vengono memorizzate senza sovrascrivere il clock locale.

### Monotonicità del clock

- *clock non-monotono* ( $C \rightarrow M$ ): non viene mantenuto un ordinamento tra tempo locale e globale, per cui si potrebbe avere una non corretta collocazione di eventi.
- *clock monotono* ( $CM$ ): mantiene l'ordinamento tra tempo locale e globale.

### Modalità di sincronizzazione

- *con timestamp* ( $T$ ): viene utilizzato un meccanismo di sincronizzazione basato su timestamp per eseguire la sincronizzazione tra nodi. I contro sono: maggior utilizzo delle risorse minor precisione dovuta al *ritardo di trasmissione*.
- *senza timestamp* ( $\neg T$ ): sfrutta l'invio/ricezione del pacchetto per sincronizzarsi.

La figura 2.3.1 riassume la tassonomia.

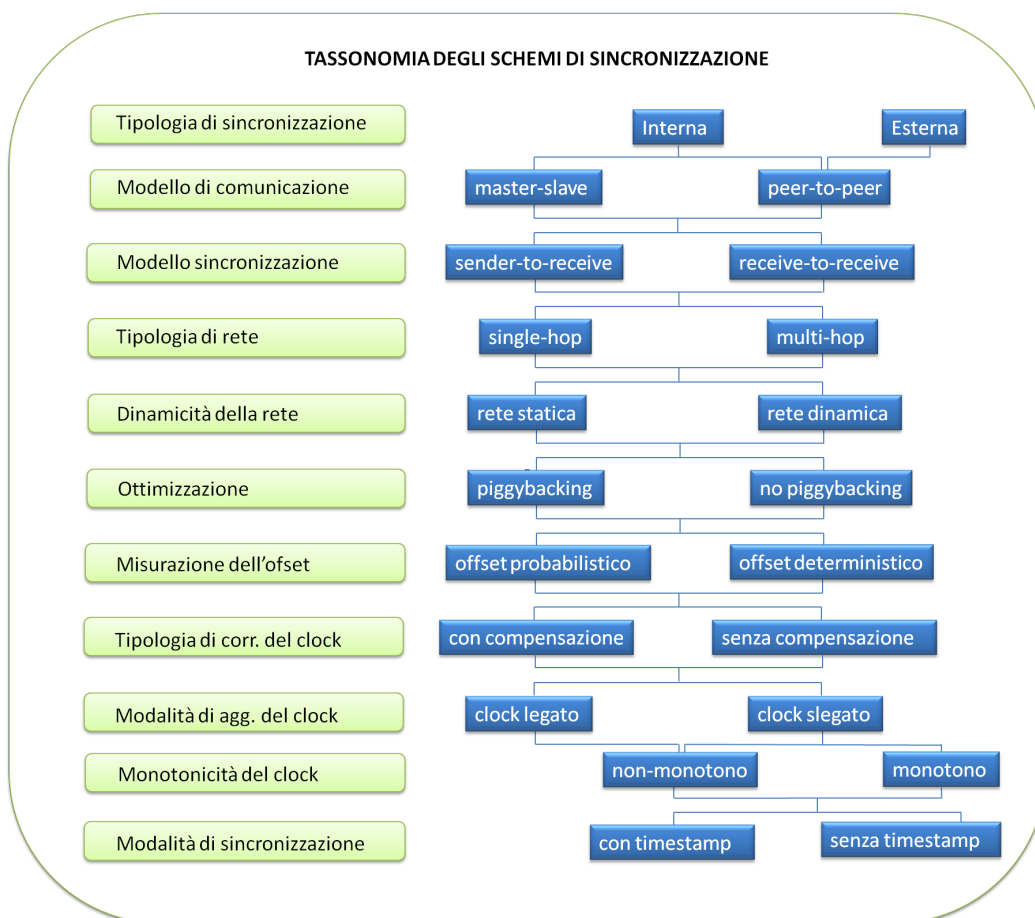


Figura 2.3.1: Tassonomia degli schemi di sincronizzazione in sintesi.

## 2.4 Rassegna della letteratura

La sincronizzazione temporale in ambito distribuito è una problematica molto sentita nella comunità scientifica e pertanto negli anni diverse soluzioni sono state proposte. Uno dei primi, più diffusi ed avanzati algoritmi in materia è Network Time Protocol (NTP), ideato da Mills[5, 8] nei primi anni Ottanta ed utilizzato per sincronizzare una rete di computer. In NTP i nodi della rete sono sincronizzati ad una sorgente di tempo globale, che è iniettata nella rete attraverso un sottogruppo di nodi master (chiamato “stratum-1” server). Questi nodi master sono sincronizzati direttamente ad una sorgente esterna di tempo come ad esempio un dispositivo GPS (tali dispositivi formano lo “stratum-0”). Tutta la rete è strutturata gerarchicamente a livelli.

Nel 2002 venne introdotto il protocollo IEEE 1588 meglio conosciuto come Precision Time Protocol (PTP)[9] ideato per sistemi locali che richiedono un'accuratezza superiore a quella ottenibile con NTP.

Tuttavia molte delle caratteristiche di una WSN precludono l'utilizzo delle esistenti tecniche di sincronizzazione in questo dominio, dove la problematica della sincronizzazione è resa più ardua per questioni quali la ridotta potenza computazionale dei nodi, il power, l'assenza di moduli particolari come ad esempio il gps, la dinamicità della rete ecc... .

Di seguito verranno illustrate le caratteristiche essenziali dei protocolli più rappresentativi di sincronizzazione per WSN.

### 2.4.1 Reference Broadcast Synchronization: RBS

Uno dei primi algoritmi ideati in materia è stato proposto da Elson, Girod e Estrin[10]. In RBS ogni nodo è normalmente non sincronizzato con il resto della rete. Un nodo segnalatore (beacon node) periodicamente invia in modalità broadcast dei pacchetti di riferimento detti *beacon* ai nodi all'interno del suo raggio di trasmissione. L'evento di ricezione del pacchetto beacon è utilizzato come riferimento di tempo per la stima di offset e skew dei nodi vicini. Quando un nodo riceve un pacchetto beacon registra il tempo di arrivo del pacchetto effettuando il timestamp in accordo al proprio clock locale. In seguito scambia il proprio timestamp con il resto dei nodi. Ogni nodo utilizza i pacchetti contenenti i timestamp e la regressione lineare per stimare gli offset e drift relativi ai nodi vicini. La cosa interessante di RBS è che registra il timestamp solo nei nodi ricevitori, infatti non interessa il tempo in cui il nodo di riferimento invia il pacchetto beacon. Per cui tutti gli errori di incertezza dal lato trasmettitore sono eliminati. Questa caratte-

ristica rende l'algoritmo particolarmente adatto a tutti quei dispositivi hardware che non forniscono un timestamp a livello MAC. Il meccanismo proposto con RBS può essere facilmente esteso a multi-hop network. Lo svantaggio principale di RBS risulta nel fatto che non si verifica direttamente una sincronizzazione tra nodo trasmettitore e ricevitore dei pacchetti beacon. Inoltre un numero elevato di scambi di messaggi è richiesto per raggiungere la sincronizzazione.

### 2.4.2 Timing-sync Protocol for Sensor Network: TPSN

Ganeriwat et al. [11] proposero un loro protocollo di sincronizzazione con un approccio innovativo del tipo transmitter-receiver. TPSN è un algoritmo gerarchico suddiviso in due fasi: la prima chiamata *level discovery phase* e la seconda *synchronization phase*. Lo scopo della prima fase è quello di costituire una topologia gerarchica della rete, strutturata in livelli. In questa fase ad ogni nodo viene assegnato un livello. Solo ad un nodo viene assegnato il livello 0, questo nodo viene chiamato nodo radice (root node). Nella seconda fase avviene la sincronizzazione della rete secondo la seguente procedura: un nodo del livello  $i$  si sincronizza ad un nodo di livello  $i-1$ . Alla fine della fase di sincronizzazione, tutti i nodi sono sincronizzati al nodo radice ed in questo modo si raggiunge la sincronizzazione globale della rete.

### 2.4.3 Delay Measurement Time Synchronization for WSNs: DMTS

In DMTS [12] un leader è selezionato come nodo reference e trasmette il suo clock. Tutti i dispositivi che ricevono il timestamp inviato dal nodo reference, correggono il clock con il tempo appena ricevuto aggiungendoci il ritardo di trasmissione. Progettato per avere un basso overhead sulla rete e un basso consumo energetico. L'accuratezza dipende evidentemente dalla misura del ritardo di trasmissione.

### 2.4.4 Flooding Synchronization Time Protocol: FTSP

FTSP [13] può essere utilizzato per sincronizzare un'intera rete e necessita che ogni nodo venga identificato univocamente con un codice ID. Il nodo con ID più basso è eletto nodo radice e serve come sorgente di riferimento del tempo globale della rete. Se questo nodo viene a mancare, il nodo con ID più basso nella rete rimanente viene eletto nuovo nodo radice. Il nodo radice periodicamente inoltra in modalità broadcast un messaggio di sincronizzazione contenente il relativo timestamp di invio. I nodi che non hanno ancora ricevuto questo messaggio, ne registrano il

timestamp contenuto ed il tempo di arrivo in accordo con la loro stima del tempo globale. In seguito aggiornano la propria stima del tempo globale e inoltrano il messaggio ai nodi vicini. Il timestamp è realizzato a livello MAC per minimizzarne i ritardi variabili dovuti ai livelli superiori. Ciascun nodo colleziona otto coppie di timestamp, tempo di arrivo dei messaggi di sincronizzazione e utilizza la regressione lineare di questi punti per stimare la differenza di offset e skew con il nodo radice.

### 2.4.5 Lightweight Tree-based Synchronization: LTS

LTS [14] è una variazione del protocollo TPSN [11] in cui viene sacrificata la precisione della sincronizzazione per guadagnare in consumo di potenza. Diversamente da molte altre tecniche che tendono alla massima precisione, in LTS le necessità di comunicazione e calcolo per la sincronizzazione del singolo nodo sono state significativamente ridotte traendo vantaggio dal rilassamento del vincolo di accuratezza. Lo schema proposto sacrifica l'accuratezza eseguendo le operazioni di sincronizzazione con minor frequenza e solo tra alcuni nodi. Sono proposti due algoritmi: entrambi assumono l'esistenza di almeno un nodo che abbia accesso ad una sorgente di riferimento di tempo globale.

Nel primo algoritmo si usa un approccio centralizzato multi-hop dove gli aggiornamenti periodici e la sincronizzazione sono gestiti da un nodo radice. La base dell'algoritmo è la costruzione di un albero di coperture (spanning tree)  $\mathcal{T}$  a profondità minima che comprende i nodi della rete. Il nodo radice inizia l'algoritmo sincronizzandosi con tutti i suoi figli in  $\mathcal{T}$  (single-hop) sfruttando il round-trip time<sup>2</sup> di un messaggio. Successivamente ogni figlio del nodo radice si sincronizza con i propri figli. Questo processo continua finché i nodi foglia di  $\mathcal{T}$  sono stati sincronizzati. Il tempo necessario per un ciclo completo dell'algoritmo è proporzionale alla profondità di  $\mathcal{T}$ . Un nuovo spanning-tree  $\mathcal{T}$  è costruito ogni volta che l'algoritmo viene eseguito. La frequenza di sincronizzazione è calcolata in base alla precisione richiesta, alla profondità dell'albero  $\mathcal{T}$  e al limite di deriva del clock.

Nell'algoritmo decentralizzato multi-hop si usa lo schema distribuito dove i singoli nodi sono responsabili per iniziare ad eseguire la re-sincronizzazione. Nessun tipo di spanning-tree è necessario per dirigere la comunicazioni tra nodi. Quando un nodo necessita di esser sincronizzato, spedisce un messaggio di richiesta al nodo di riferimento più vicino utilizzando un qualsiasi algoritmo di routing. La frequenza di sincronizzazione è calcolata in base all'accuratezza richiesta, alla distanza in numero di hop dal nodo di riferimento ed al drift del clock.

---

<sup>2</sup>E' una misura del tempo impiegato da un pacchetto di dimensione trascurabile per viaggiare da un nodo della rete ad un altro e tornare indietro.

### 2.4.6 Tiny-Sync e Mini-Sync (TS/MS)

Tiny-Sync e Mini-Sync [15] sono entrambi algoritmi di sincronizzazione che sfruttano la trasmissione di informazioni tra coppie di nodi (pairwise synchronization). Gli autori assumono che ciascun clock può essere approssimato con il seguente modello a frequenza costante:

$$t_i(t) = a_i t + b_i \quad (2.4.1)$$

dove  $a_i$  e  $b_i$  sono drift e offset del clock del nodo  $i$ . Considerando due nodi 1 e 2 con i loro rispettivi clock hardware  $t_1(t)$  e  $t_2(t)$  dalla 2.4.1 segue che  $t_1(t)$  e  $t_2(t)$  sono linearmente legati da:

$$t_1(t) = a_{12} t + b_{12} \quad (2.4.2)$$

dove  $a_{12}$  e  $b_{12}$  rappresentano il drift relativo e l'offset relativo tra i due nodi.

Entrambi gli algoritmi usano valori di round-trip time per ottenere stime di offset e drift tra coppie di nodi. La raccolta di tali informazioni (data collection) avviene nel seguente modo: il nodo 1 manda al nodo 2 un messaggio di “probe” e registra l'istante di tale invio corrispondente al tempo  $t_0$ . Quando il nodo 2 riceve il messaggio genera il timestamp  $t_b$  in accordo col proprio clock locale e immediatamente spedisce indietro al nodo 1 un messaggio di risposta contenente tale valore. Alla ricezione della risposta del nodo 2, il nodo 1 registra il timestamp  $t_r$ . Sfruttando l'ordine degli eventi descritti (corrispondenti alla ricezione/trasmissione di un messaggio), i relativi timestamp e la relazione 2.4.2 è possibile ricavare il seguente sistema di disuguaglianze:

$$\begin{cases} t_0 < a_{12} t_b + b_{12} \\ t_r > a_{12} t_b + b_{12} \end{cases} \quad (2.4.3)$$

I tre valori dei timestamp ( $t_0, t_b, t_r$ ) formano un *data point*. Entrambi gli algoritmi lavorano con un gruppo di data point collezionati con lo scambio precedentemente spiegato e permettono di fare un'analisi di drift e offset come rappresentato in figura 2.4.1.

Le linee tratteggiate in figura 2.4.1 soddisfano l'equazione 2.4.2 e rappresentano i vincoli imposti dai data point. Una è la retta con pendenza maggiore e offset minore ( $\overline{a_{12}}$  e  $\underline{b_{12}}$ ) mentre l'altra è la retta con pendenza minore e offset maggiore ( $\underline{a_{12}}$  e  $\overline{b_{12}}$ ). Il drift relativo e l'offset relativo di una coppia di nodi sono limitati dalle seguenti disuguaglianze:

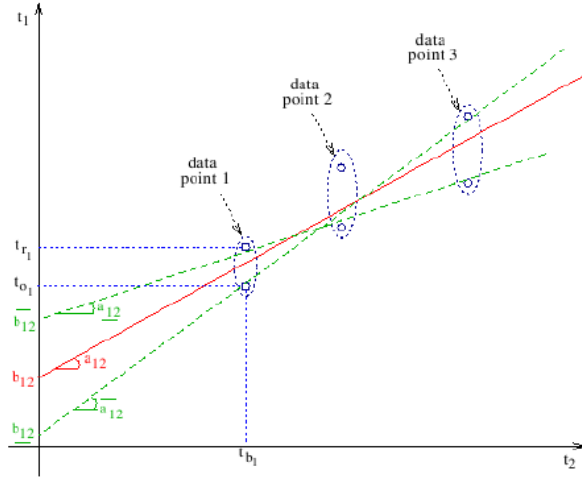


Figura 2.4.1: Dipendenze lineari e vincoli imposti ad  $a_{12}$  e  $b_{12}$  con tre data point.

$$\begin{cases} \underline{a}_{12} \leq a_{12} \leq \overline{a}_{12} \\ \underline{b}_{12} \leq b_{12} \leq \overline{b}_{12} \end{cases} \quad (2.4.4)$$

Possiamo dire che più vicini sono i limiti ( $[\overline{a}_{12}$  e  $\underline{a}_{12}]$   $[\overline{b}_{12}$  e  $\underline{b}_{12}]$ ), più la stima di offset e drift è buona e la precisione della sincronizzazione è maggiore. In questo modo non possono essere determinati esattamente i valori di offset e drift, ma possono essere ben stimati, e la stima è tanto migliore quanto maggiore è il numero di data point collezionati da un nodo. Ovviamente i limiti di memoria e di computazione dei dispositivi limitano il numero di data point disponibili per nodo. Per questo TinySync e MiniSync sfruttano il fatto che non tutti i data point sono utili per la stima. Come possiamo vedere in figura 2.4.1 il data point 2 non ha nessuna influenza sulla stima dei parametri delle disuguaglianze precedenti. Questa analisi è sfruttata in TinySync il quale mantiene in memoria solamente i due data point che danno i migliori bound, tuttavia la stima non è sempre la migliore possibile. MiniSync estende TinySync, infatti esso cerca la soluzione ottima con relativo aumento della complessità dell'algoritmo.

### 2.4.7 Scalable Lightweight Time Synchronization Protocol for WSNs: SLTP

SLTP [2] è orientato al power ed usa la regressione lineare su collezioni di dati per sincronizzare i nodi. E' costituito da due fasi, una prima fase di configurazione per reti statiche e dinamiche in cui viene eletto il gruppo leader e i membri. Una



seconda fase dedicata alla sincronizzazione dei membri della rete.

### 2.4.8 Reference Based, Tree Structured Time Synchronization: TSRT

TSRT [16] si pone come obiettivo principale quello di minimizzare la complessità di sincronizzazione ponendo come vincolo l'accuratezza di sincronizzazione. Lavora in due fasi: una prima fase costruisce la struttura ad albero topologica della rete e la seconda sincronizza i clock locali e valuta lo stato della rete. Prevede la presenza di un nodo reference (oppure multipli) che periodicamente trasmette un beacon ai suoi vicini. A differenza di altri protocolli questo approccio sfrutta la comunicazione multicanale per migliorare la precisione e minimizzare l'overhead di comunicazione e di conseguenza il consumo di energia. In sostanza per evitare collisioni con i nodi vicini (ossia allo stesso hop) nodi dello stesso hop comunicano su canali differenti. Di base utilizza la regressione lineare per la correzione dei clock locali.

### 2.4.9 Feedback Based Synchronization: FBS

FBS [17] è uno schema di sincronizzazione che si differenzia dai precedenti per l'approccio basato sulla teoria del controllo. A differenza dei precedenti schemi menzionati che eseguono generalmente una compensazione della variazione del clock sul breve periodo modellando il clock come lineare, FBS considera le perturbazioni del clock causate da variazioni di lungo periodo che generalmente gli altri schemi stimano con metodi statistici [10, 18, 19]. Questi solitamente richiedono una grande quantità di dati e potenza computazionale, riflettendosi su consumo di tempo, memoria e risorse che potrebbero andare ben oltre le capacità di un nodo sensore. L'idea che c'è alla base è quella di usare un controllore proporzionale integrale (PI) in anello chiuso per compensare offset/skew del clock locale. Può essere usato anche in multi-hop, in questo caso è necessaria una prima fase, dove viene costruito un albero di coperture (spanning tree)  $\mathcal{T}$  a profondità minima che comprende i nodi della rete.

Concludiamo questa sezione riepilogando con la tabella 2.1 le caratteristiche dei protocolli appena presentati. Si fa presente che pur non avendo ancora descritto FLOPSYNC esso è presente in tabella, in quanto si vuol dare al lettore un'idea della tipologia di schema proposto in relazioni a quelli esistenti.

	I vs E	P-P vs M-S	S-R vs R-R	S-H vs M-H	RS vs RD	P vs $\neg$ P	OP vs OD	CC vs C-C	CS vs CL	C-M vs CM	T vs $\neg$ T
RBS ('02)	~	P-P	R-R	S-H	RS	$\neg$ P	OD	CC	CS	C-M	T
TPSN ('03)	~	M-S	S-R	M-H	RS	$\neg$ P	OP	C-C	CL	C-M	T
DMTS ('03)	~	M-S	S-R	M-H	RS	P	OD	C-C	CL	C-M	T
FTSP ('04)	~	M-S	S-R	M-H	RD	P	OP	CC	CL	C-M	T
LTS ('03)	~	M-S	S-R	M-H	RD	$\neg$ P	OD	C-C	CL	C-M	T
TS/MS ( '03)	I	P-P	S-R	M-H	RS	$\neg$ P	OP	CC	CL	C-M	T
SLTP ('07)	~	M-S	S-R	M-H	RD	$\neg$ P	OP	CC	CL	C-M	T
TSRT ('11)	~	M-S	S-R	M-H	RD	$\neg$ P	OD	CC	CL	C-M	T
FBS ('10)	E	M-S	S-R	M-H	RD	$\neg$ P	OP	CC	CL	C-M	T
FLOPSYNC ( '13)	E	M-S	S-R	M-H	RD	P	OP	CC	CS	CM	$\neg$ T

Tabella 2.1: Classificazione degli schemi di sincronizzazione

## 2.5 Valutazione dei protocolli di sincronizzazione

Non è possibile affermare in modo assoluto che uno schema sia nettamente migliore di un altro in ogni possibile applicazione di reti wireless. Piuttosto, è molto probabile che la scelta di un protocollo sarà guidata dalle caratteristiche ed esigenze d'impiego. Per esempio, un protocollo a bassa precisione potrebbe essere appropriato per molte applicazioni di monitoraggio ambientale. Tuttavia, lo stesso protocollo potrebbe essere inusabile in applicazioni come data fusion<sup>3</sup>, TDMA<sup>4</sup> schedules, ecc....

Valutare la bontà di uno schema di sincronizzazione per WSN è un compito piuttosto arduo. Un esempio è rappresentato da quanto accaduto in passato agli autori di TPSN [11] implementando l'algoritmo proposto da RBS [10] per operare un confronto con il loro. Incredibilmente lo stesso algoritmo implementato sulla stessa piattaforma dagli autori TPSN ha raggiunto un errore medio nel caso di single hop di  $29.1\mu s$  contro gli  $11\mu s$  dichiarati da RBS. TPSN pur avendo un errore di  $16.9\mu s$  vinceva il confronto in questa valutazione.

Questo piccolo aneddoto mostra la necessità di definire delle metriche oggettive per la valutazione dei protocolli di sincronizzazione per WSN. Per motivi di chiarezza si dividono i criteri in due classi: hardware-dependent ed hardware-independent. I primi includono precisione di sincronizzazione e consumo energe-

<sup>3</sup>E' il processo di integrazione di molteplici dati e conoscenze che rappresentano lo stesso oggetto del mondo reale in una rappresentazione coerente, precisa e utile.

<sup>4</sup>Time Division Multiple Access è una tecnica di multiploazione numerica in cui la condivisione del canale è realizzata mediante ripartizione del tempo di accesso allo stesso.

tico. I restanti comprendono complessità computazionale, tempo di convergenza, scalabilità e tolleranza ai guasti. Nel loro insieme, queste misure forniscono una buona caratterizzazione dell'applicabilità e le prestazioni di ciascun protocollo.

### 2.5.1 Valutazione hardware-dependent

Questi criteri di valutazione sono fortemente legati all'hardware su cui si decide di implementare lo schema di sincronizzazione, pertanto non è semplice valutare quantitativamente schemi implementati su hardware differenti.

**Accuratezza di sincronizzazione:** è un valore strettamente legato all'errore di sincronizzazione. Mentre l'errore di sincronizzazione è una funzione del tempo  $t$  e si riferisce a ciascun nodo, la precisione della sincronizzazione è un parametro globale riferito a tutta la rete di nodi definito come valore massimo oppure valore medio della precisione istantanea in un intervallo di tempo. Possiamo definire la precisione istantanea  $p(t)$ , altrimenti nominata come dispersione di gruppo (o group dispersion) in [10], come l'errore massimo di fase tra una qualsiasi coppia di nodi nel seguente modo:

$$p(t) = \max_{ij} \{t_i - t_j\} \quad (2.5.1)$$

Se per la sincronizzazione ci si riferisce ad una sorgente di riferimento, come può esserlo il nodo reference, allora la precisione di sincronizzazione è data da:

$$p(t) = \max_i \{t_i\} \quad (2.5.2)$$

**Efficienza energetica:** è tra le metriche che hanno più peso nella maggior parte delle applicazioni di reti di sensori. Le ragioni che inducono a richiedere un basso consumo energetico sono legate essenzialmente alla concezione di nodo sensore ed al suo utilizzo. Si ricorda che nella gran parte delle applicazioni ogni sensore ha una riserva d'energia limitata e non rinnovabile e una volta messo in opera, deve lavorare autonomamente; per questo motivo tali dispositivi devono mantenere costantemente i consumi molto bassi in modo da avere un maggior ciclo di vita. In condizioni di lavoro realistiche infatti, è impensabile sostituire le pile di un nodo sensore, sia per motivi di costo che di accessibilità dei nodi stessi, in quanto potenzialmente disposti in ambienti ostili e difficilmente accessibili. La ricarica in loco, sfruttando l'energia presente nell'ambiente, solitamente permette solo bassi livelli di potenza e non è sempre praticabile.

## 2.5.2 Valutazione hardware-independent

Le seguenti metriche sono facilmente confrontabili tra vari schemi di sincronizzazione indipendentemente dall'hardware su cui sono state implementate.

**Complessità computazionale:** esprime la quantità di risorse di calcolo (spazio e tempo) necessarie per eseguire lo schema di sincronizzazione e il numero di messaggi scambiati dal protocollo per eseguire la sincronizzazione.

**Tempo di convergenza:** è il tempo necessario per sincronizzare una rete. Un protocollo che richiede lo scambio di un gran numero di messaggi per la sincronizzazione si tradurrà in un tempo di convergenza più lungo. Pertanto ridurre la complessità del messaggio è di vitale importanza per il rapporto costo-efficienza di un protocollo di sincronizzazione per WSN.

**Scalabilità:** è intesa come capacità dello schema di adattarsi alle variazioni, sia nel numero di nodi che nella posizione geografica.

**Tolleranza ai guasti:** gioca un ruolo importante in quanto il canale wireless è piuttosto soggetto ad errori. La scarsa affidabilità della consegna del messaggio in un mezzo wireless può avere effetti devastanti sul protocollo di sincronizzazione perché essa richiede scambi di più messaggi. Sono comprese in questa metrica anche le questioni riguardanti il guasto del nodo reference in tutti quei protocolli che basano lo schema di sincronizzazione su quest'ultimo.

La tabella 2.2 compara i protocolli finora presentati con i criteri appena stabiliti. Nella colonna della complessità i simboli hanno il seguente significato:

- $n$ , il numero di nodi della rete;
- $m$ , il numero di sincronizzazioni del nodo;
- $k$  è una costante che rappresenta il numero di campioni utilizzati nella regressione lineare per la stima dell'errore futuro;
- $v$ , indica il numero di nodi considerati vicini per un dato nodo dove i vicini vengono definiti utilizzando l'algoritmo di spanning tree.

Come si può notare dalla comparazione, FLOPSYNC ha delle ottime performance rispetto agli altri e le motivazioni verranno spiegate nel corso di questo lavoro. Si noti anche come tutti i protocolli, eccetto FBS e FLOPSYN, che privilegiano la complessità trascurano l'accuratezza. Inoltre quando si parla di accuratezza è bene considerare sia quella un'istante dopo l'evento di sincronizzazione che quella un'istante prima, altrimenti l'informazione potrebbe essere forviante.

	Accuratezza		Eff. ener.	Complessità			Tempo Conv.	Scal.	Toll. guasti
	$t(k)^+$	$t(k+1)^-$		Spaziale	Temporale	Canale			
RBS ('02)	alta	alta	alta	$O(n)$	$O(mn^2)$	$m + mn$	molto alto	molto bassa	n/a
TPSN ('03)	alta	bassa	alta	$O(1)$	$4m(n-1)$	$m + mn$	n/a	bassa	n/a
DMTS ('03)	alta	bassa	molto alta	$O(1)$	$O(1)$	$O(m)$	alto	bassa	n/a
FTSP ('04)	alta	alta	alta	$O(k)$	$2mn$	$O(mn)$	basso	media	media
LTS ('03)	media	bassa	bassa	$O(1)$	$4n - 2$	$4v$	molto alto	media	media
TS ('03)	alta	alta	alta	$O(k)$	$4m(n-1)$	$m + mn$	alto	bassa	n/a
SLTP ('07)	alta	alta	alta	$O(k)$	$O(m)$	$Cm$	medio	alta	alta
TSRT ('11)	alta	alta	bassa	$O(k)$	$O(m)$	$O(mn)$	alto	media	buona
FBS ('10)	molto alta	molto alta	molto alta	$O(1)$	$O(1)$	$O(m)$	basso	media	media
FLOPSYNC ('13)	molto alta	molto alta	molto alta	$O(1)$	$O(1)$	$O(m)$	molto basso	alta	buona

Tabella 2.2: Comparazione delle performance degli schemi di sincronizzazione. Per la complessità,  $n$  indica il numero di nodi,  $m$  il numero di sincronizzazioni,  $C$  il numero di *cluster head* [2],  $v$  il numero di vicini dati dall'algoritmo spanning-tree e  $k$  è una costante che indica il numero di campioni memorizzati per la regressione lineare.

## 2.6 Motivazione della ricerca

Tutti gli schemi di sincronizzazione esistenti ad oggi per WSN (lo si può notare anche dai più esemplificativi presentati in questo capitolo) sfruttano l'invio di timestamp per sincronizzare i clock, ma è abbastanza evidente che il suo utilizzo, eccetto nella fase di boot, è del tutto inutile. La sua informazione è intrinsecamente contenuta nell'arrivo del pacchetto di sincronizzazione, pertanto si potrebbe pensare di rimuovere il timestamp da uno schema di sincronizzazione. La rimozione del timestamp porta con se diversi benefici, si pensi ad esempio a come questo cambiamento si riflette sul *ritardo di trasmissione* piuttosto che sul *power*.

Un'altra questione importante riguarda la necessità di un meccanismo di compensazione skew/drift.

La figura 2.6.1 riporta il risultato di un test fatto su un hardware reale (descritto nel Capitolo 4) che consiste nel tenere sincronizzato il clock di due nodi di una WSN solo tramite sincronizzazione del clock, questo significa inoltrando ogni minuto il timestamp del nodo reference. L'istante successivo alla sincronizzazione, ad esempio  $e_1(t(k)^+)$ , l'errore è ridotto a circa  $1\mu s$ . E' noto che la frequenza degli oscillatori dell'esperimento differisce di circa  $20ppm$ , e infatti come si può notare

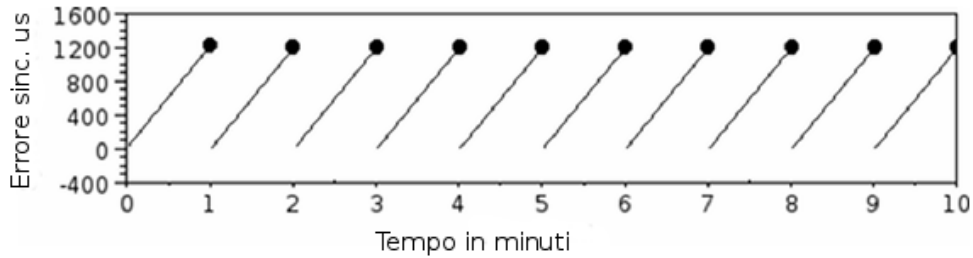


Figura 2.6.1: Errore di sincronizzazione in assenza di meccanismo di compensazione.

dalla figura 2.2.1 l'errore un'istante prima della sincronizzazione (rappresentato dai punti,  $e_1(t(k+1)^-)$ ) è di circa  $1.82ms$ . Sebbene sia possibile ridurre questo errore attraverso una sincronizzazione più frequente, l'unico modo per tenerlo il più possibile vicino ad  $e_1(t(k)^+)$  è quello di applicare un meccanismo di compensazione dello skew. Per quanto riguarda la monotonia, le correzioni di clock eseguite possono causare salti all'indietro nel tempo locale, con un'entità nel caso peggiore di  $1.182ms$  ogni minuto.

Si noti che, sebbene la compensazione dello skew riduca l'entità dei salti del clock, questi sono ancora presenti. Ad esempio, drift causati da variazioni di temperatura, o jitter causati da instabilità di frequenza di breve periodo e la non linearità dell'oscillatore [17], possono causare una differenza tra *skew atteso* e *skew attuale*, ed il risultato è ancora il manifestarsi di un salto con la sincronizzazione successiva. Quindi, l'unica soluzione robusta è quella di escogitare uno schema di sincronizzazione e di compensazione che garantisca tempi locali monotoni per progettazione.

Sarebbe interessante inoltre, avere la possibilità di poter regolare lo schema di sincronizzazione in base al tipo di applicazione e alle modalità in cui la WSN andrà ad operare, fornendo all'utilizzatore la garanzia che lo schema operi entro certi limiti.

E' stata la mancanza di questi aspetti ad indurre la ricerca del presente lavoro accompagnata da una buona dose di pura e sana curiosità scientifica.

# Capitolo 3

## FLOPSYNC

Questo capitolo è quello che possiamo definire il cuore pulsante dell'intera trattazione. Si basa su una prospettiva innovativa di progettazione delle componenti di un sistema operativo che vengono concepite fin dall'inizio come problema di controllo [20], non invece create e successivamente dotate di controllo. Come sarà mostrato, adottare quest'atteggiamento significa trattare un sistema informatico alla stregua di uno fisico ottenendo un guadagno davvero significativo.

### 3.1 Approccio proposto

L'approccio proposto richiede una prospettiva diversa del problema globale. Si potrebbe infatti dire che praticamente qualsiasi lavoro precedente in materia si colloca in un scenario strutturato, semplificato per brevità, come segue:

1. prima fase, sincronizzare i clock della rete di sensori attraverso un meccanismo basato sull'invio di timestamp aggiornando il clock del nodo da sincronizzare in maniera *slegata* o *legata* che sia come discusso nella sezione 2.3;
2. seconda fase, con il clock sincronizzato, stimare lo skew del clock locale, così da fornire primitive per la corretta temporizzazione tra un evento di sincronizzazione e l'altro.

Qui si mostra qualcosa di diverso. In primo luogo, nella sezione 3.2, si modella il problema della sincronizzazione limitato ad uno scenario piuttosto semplificato. Questo problema sarà risolto interamente con la teoria del controllo ad anello chiuso con l'assunto che non esista alcun altro componente eccetto che un efficiente sistema di flooding. Per risolvere il problema base della sincronizzazione verrà fornita una struttura formale con l'idea di separare sintesi del controllore e modellazione del disturbo. Verrà modellato pertanto il disturbo e dalla sua diretta conoscenza ne

scaturirà il controllore atto a rigettarlo. Come diretta conseguenza delle analisi di cui sopra, verrà mostrato che, tentare di neutralizzare drift termici mediante compensazione in anello aperto attraverso la misurazione della temperatura, in generale non fornisce risultati affidabili e robusti e soprattutto non ripaga il costo sostenuto in termini di complessità dell’algoritmo di controllo e difficoltà di messa a punto. Verrà studiata invece, una metodologia per tarare i parametri del controllore impiegando ancora una volta solo i dati nominali e le condizione operative previste.

Poi, nella sezione 3.3, verrà analizzata retrospettivamente la soluzione proposta e si formalizzerà una soluzione che porti ad uno schema di sincronizzazione capace di compensare variazione di skew/drif con *clock monotono*.

Infine, nella sezione 3.4, si passa dallo schema di sincronizzazione al protocollo vero e proprio, rilassando le ipotesi semplificatrici ed entrando quindi nel mondo reale, dove questioni come integrazione nello stack radio piuttosto che ottimizzazione di potenza verranno affrontate e discusse.

## 3.2 La sincronizzazione come problema di controllo

Si consideri una WSN composta da  $N$  nodi. Uno di loro invia uno speciale pacchetto detto *bsp* (Broadcast Synchronization Packet) il cui contenuto è irrilevante, e con un periodo fissato pari a  $T$  conosciuto da tutti gli altri. Il nodo che invia è il nodo reference, pertanto il suo clock si assume come riferimento e il suo tempo  $t$  è il tempo globale (ai fini della trattazione è irrilevante che il clock del nodo reference sia “esatto” o meno, l’importante è assumere il suo tempo come “vero”). La trasmissione del *bsp* avviene tramite uno schema di flooding che bypassa completamente il protocollo di rete in modo da poter assumere che non vi siano contese di accesso al canale che introducono incertezza nel tempo di trasmissione. Inoltre si assume che, ogni nodo è in grado di ricevere un pacchetto in qualsiasi momento ed ogni pacchetto trasmesso è correttamente ricevuto da tutti i nodi della rete (idealità del mezzo trasmissivo). L’obiettivo è avere ogni nodo **sincronizzato** col nodo reference, dove per “sincronizzato” si intende che l’errore tra tempo locale e tempo globale sia nullo (o quasi).

La soluzione proposta per il problema base è quella di dotare ogni nodo di un controllore retroazionato capace di sincronizzare il clock locale semplicemente basandosi sulla misurazione della discrepanza tra *tempo di arrivo atteso* e *tempo di arrivo effettivo* del *bsp*, senza operare correzioni sul clock hardware del nodo.



### 3.2.1 Modello del sistema da controllare

Prima di progettare la legge di controllo è necessario fornire un modello dinamico del sistema da controllare.

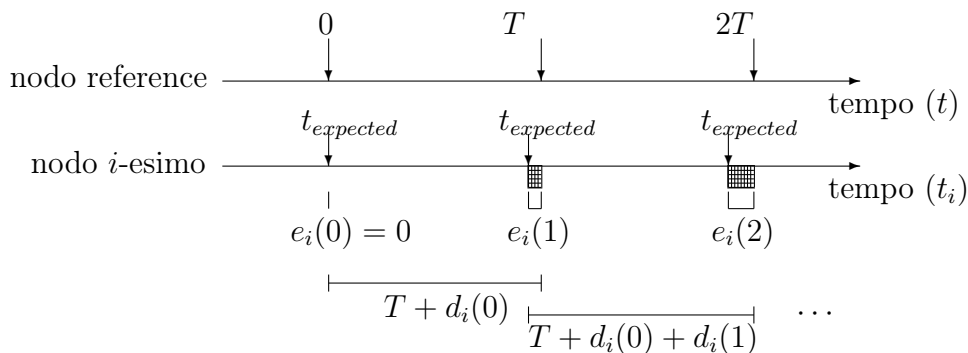


Figura 3.2.1: Sistema da controllare nel problema base.

A tal fine, si consideri la situazione schematizzata in figura 3.2.1. Il lettore supponga che ad un certo istante  $k$  l' $i$ -esimo nodo sia sincronizzato con quello reference, ossia che all'istante  $k$  il tempo di arrivo atteso (*expected*) del bsp  $t_i^e(k)$  e quello effettivo (*actual*)  $t_i^a(k)$ , entrambi contanti col clock del nodo locale, coincidano (questa è solo un'ipotesi tecnica per spiegare il modello). Allora il nodo reference trascorso un periodo di tempo  $T$  invierà il bsp  $(k+1)$ -esimo. Quando il bsp verrà ricevuto dal nodo  $i$ -esimo, quest'ultimo ne misurerà il tempo *actual* ad esempio misurando  $t_i^a(k+1)$  (istante d'inizio del bsp). Il sistema non controllato quindi, è descritto dalle equazioni del modello dinamico lineare tempo invariante a tempo discreto seguenti:

$$\begin{cases} t_i^a(k+1) = t_i^a(k) + T \\ t_i^e(k+1) = t_i^e(k) + T \\ e_i(k) = t_i^e(k) - t_i^a(k) \end{cases} \quad (3.2.1)$$

dove  $e_i$  esprime l'errore definito nella sezione 2.2 come la differenza tra il tempo *expected* ed *actual* del bsp.

Si conosce inoltre che, sul sistema appena modellato, agisce un disturbo  $d_i$  le cui cause sono legate a molteplici contributi di cui al momento se ne ignorino le origini. Aggiungendo questa informazione al modello poc'anzi definito diventa:

$$\begin{cases} t_i^a(k+1) = t_i^a(k) + T \\ t_i^e(k+1) = t_i^e(k) + T + \mathbf{d}_i(\mathbf{k}) \\ e_i(k) = t_i^e(k) - t_i^a(k) \end{cases} \quad (3.2.2)$$

Con semplici passaggi algebrici si arriva al sistema semplificato seguente:

$$e_i(k+1) = t_i^e(k+1) - t_i^a(k+1) \quad (3.2.3)$$

$$= t_i^e(k) - t_i^a(k) + d_i(k) \quad (3.2.4)$$

$$= e_i(k) + d_i(k). \quad (3.2.5)$$

Indicando con  $z$  la variabile complessa della trasformata  $\mathcal{Z}$  e indicando con la corrispondente lettera maiuscola la trasformata  $\mathcal{Z}$  del segnale corrispondente, il modello matematico 3.2.5 può essere scritto in forma di funzione di trasferimento come:

$$E_i(z) = P_i(z)D_i(z) \quad (3.2.6)$$

dove la trasformata  $\mathcal{Z}$  del processo  $P$  da controllare si ottiene facilmente dalla 3.2.5 ed è pari a:

$$P_i(z) = \frac{1}{z-1}. \quad (3.2.7)$$

Il polo  $z=1$  evidenzia la natura integrativa del sistema, in accordo con l'accumularsi di  $d_i(k)$  osservabile in figura 3.2.1, da qui la necessità di retroazione per raggiungere la stabilità e le proprietà di convergenza dell'errore desiderate. Si fa notare che  $P_i(z)$  può essere anche scritta nella forma equivalente:

$$P_i(z) = \frac{1}{1-z^{-1}} \quad (3.2.8)$$

nota anche come proprietà di accumulazione e mostra la ragione della natura integrativa del sistema descritto come la somma del corrente disturbo ai precedenti prodotti dal ritardo di un passo dell'anello di retroazione. Pertanto il modello 3.2.6 rappresenta correttamente il comportamento ad anello aperto del processo, consentendo così un problema ben posto di sintesi del controllore in retroazione. Si noti che lo stesso non sarebbe vero se il tempo di attesa del prossimo bsp fosse basato sul tempo di arrivo actual invece che expected.

### 3.2.2 Analisi del disturbo e modellazione

Nella precedente sezione è stato introdotto il disturbo  $d_i$  ed è stato detto essere legato alla somma di più fattori. Di seguito si cerca di denotare e modellare il comportamento del disturbo.

Il disturbo è definito come:

$$d_i(t) = \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f(\tau)}{f_o} d\tau = \int_{t_i(k)}^{t_i(k+1)} \frac{f_i(\tau) - f_o}{f_o} d\tau, \quad (3.2.9)$$

dove  $f_i(\tau)$  e  $f_o$  sono gli stessi definiti nella sezione 2.2. Notare che gli estremi di integrazione dell'integrale sono espressi nel tempo locale del nodo  $t_i$  per consistenza di notazione, ma questo è sostanzialmente equivalente ad usare il tempo universale  $t$ , perché l'errore di sincronizzazione è dell'ordine dei millisecondi o meno, invece il periodo di sincronizzazione va da qualche secondo fino al minuto.

Ora si può scrivere  $f_i(t) = f_o + \delta f_{oi} + \delta f_{si}(t) + \delta f_{ji}(t)$ , dove  $\delta f_{oi}$  tiene conto delle differenze di frequenza nominali dei quarzi,  $\delta f_{si}(t)$  indica la variazione di frequenza nel tempo prodotta dalla componente di skew (e quindi tiene conto anche dei drift) e  $\delta f_{ji}(t)$  esprime le fluttuazioni di frequenza di breve periodo causate dai jitter. Per cui, sostituendo  $f_i(t)$  nella equazione 3.2.9, con semplici passaggi algebrici otteniamo

$$d_i(t) = \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f_{oi}}{f_o} d\tau + \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f_{si}(\tau)}{f_o} d\tau + \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f_{ji}(\tau)}{f_o} d\tau \quad (3.2.10)$$

Se si assume la stessa  $f_o$  per tutti i nodi per semplificare la notazione (rilassare quest'assunzione è banale), i contributi del disturbo legati a singoli termini possono essere analizzati separatamente come segue.

Il primo termine può essere riscritto come:

$$d_i^o = \frac{\delta f_{oi}}{f_o} (t_i(k+1) - t_i(k)) \quad (3.2.11)$$

che in una applicazione reale possiamo approssimare al termine costante  $\frac{\delta f_{oi}}{f_o} T$ , in quanto  $t_i(k+1) - t_i(k)$  sarà molto vicino al periodo di sincronizzazione.

Il secondo termine,  $d_i^s = \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f_{si}(\tau)}{f_o} d\tau$ , è legato essenzialmente a skew (e drift), ossia a quelle che nella sottosezione 2.1.2 sono state etichettate come variazione di breve e lungo periodo di cui le più rilevanti sono causate da fenomeni termici. A tale scopo, si assume che la caratteristica frequenza-temperatura per i

quarzi in questione (tuning fork vedere sotto sezione 2.1.1) sia quella standard di Nakazawa et al. [21], allora

$$f_i(t) = f_o \left(1 - \frac{\beta_i}{10^6} (\theta_i(t) - \theta_o(t))^2\right) \quad (3.2.12)$$

dove  $\theta_i$  è la temperatura del quarzo,  $\theta_o$  corrisponde alla temperatura della frequenza nominale (anche in questo caso si assume uguale per ogni nodo per le ragioni di cui sopra) e  $\beta_i$  il coefficiente di temperatura del quarzo espresso in  $ppm/^\circ C^2$ . Vale la pena notare che alcuni articoli aggiungono un piccolo termine lineare nella 3.2.12 e suggeriscono di tener conto anche di alcune isteresi termiche, vedere ad esempio Marchetto et al. [22]. Ad ogni modo, quasi mai i datasheet forniscono i parametri dei fenomeni menzionati. Per questo si è deciso di restare sulla 3.2.12 per ragioni di applicabilità. A fronte di questi nuovi risultati è semplice riscrivere il disturbo legato al secondo termine come:

$$d_i^s = \frac{\beta_i}{10^6} \int_{t_i(k)}^{t_i(k+1)} (\theta_i(\tau) - \theta_o(\tau))^2 d\tau. \quad (3.2.13)$$

L'ultimo termine

$$d_i^j = \int_{t_i(k)}^{t_i(k+1)} \frac{\delta f_{ji}(\tau)}{f_o} d\tau \quad (3.2.14)$$

è legato alla non linearità dell'oscillatore e a fenomeni elettrici. In generale è di modesta entità ed in ogni caso fornisce chiaramente il **limite di precisione** delle stime di tempo tra le sincronizzazioni di clock.

### 3.2.3 Sintesi del controllore

Un modo naturale per introdurre un controllo in retroazione è quello di regolare il tempo di attesa prima della prossima sincronizzazione ossia  $t_i^e(k) = T + u(k)$ , dove  $u_i(k)$  è il segnale di controllo. Notare che così facendo non è necessario nessuna modifica del clock hardware, come anticipato. Il modello del sistema controllato è quindi:

$$E_i(z) = P_i(z)(U_i(z) + D_i(z)), \quad (3.2.15)$$

dove la trasformata  $P_i(z)$  è la stessa di 3.2.8. Sulla base di questa idea la figura 3.2.2 mostra lo schema a blocchi per il controllo in retroazione del generico nodo  $i$ -esimo.

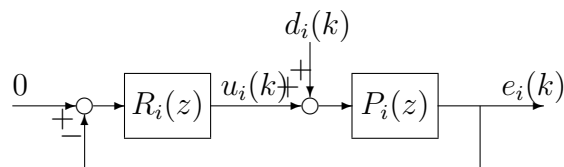


Figura 3.2.2: Schema di controllo per il nodo  $i$ -esimo.

Formalmente, detto problema viene indicato come sintesi del regolatore  $R_i(z)$  e ha come obiettivo quello di eguagliare l'uscita  $e_i(k)$  al set point, anche in presenza di un disturbo  $d_i(k)$ , agendo sull'ingresso  $u_i(k)$ . In questo caso il set point è zero perché si desidera che l'errore di sincronizzazione sia nullo.

Una tecnica di sintesi conveniente per tale problema è quella di ricavare il regolatore a partire dal calcolo della *funzione di sensitività del controllo* che, data l'indipendenza dal dispositivo, richiede solamente di caratterizzare il disturbo  $d_i$ . Si ricorda che la funzione di sensitività del controllo è la funzione di trasferimento tra l'uscita e il disturbo del sistema retroazionato, e per il problema assegnato è pari a:

$$Q_i(z) = \frac{E_i(z)}{D_i(z)} \quad (3.2.16)$$

$$= \frac{R_i(z)}{1 + L_i(z)} \quad (3.2.17)$$

$$= \frac{R_i(z)}{1 + P_i(z)R_i(z)}. \quad (3.2.18)$$

A questo punto si ha la conoscenza completa dell'intero sistema e del disturbo, quindi tutte le informazioni per sintetizzare il controllore. La progettazione del controllore può essere fatta per gradi: il primo termine del disturbo rappresentato dall'equazione 3.2.11, come spiegato nella sezione precedente può essere approssimato ad una costante e pertanto può essere eliminato dall'azione integrale di  $R_i(z)$ , per cui possiamo concentrarci solamente sui restanti due. Il contributo del disturbo legato al jitter dell'equazione 3.2.14 è di natura troppo veloce per essere contrastato in qualsiasi periodo di tempo ragionevole per la sincronizzazione, quindi basta richiedere che l'ampiezza della funzione di sensitività del controllo ad alta frequenza sia la più piccola possibile compatibilmente con gli altri requisiti, al fine di evitare un superfluo ed inutile disturbo  $u_i(k)$ . Infine c'è il contributo legato a skew, equazione 3.2.13, il più complicato e difficile disturbo da rigettare. Senza avventurarsi in una rigorosa e dettagliata trattazione matematica di  $d_i^s$  legata ai fenomeni termici che agiscono sul quarzo (che invece verrà proposta di seguito), per

la sintesi del regolatore è utile focalizzarsi solo sull'andamento generale del disturbo. Con buona approssimazione si può affermare che la variazione di temperatura del quarzo ha un andamento non troppo dissimile da quello esponenziale che parte dalla temperatura subito prima dello scalino termico e converge ad un valore asintotico per il tempo che avanza verso infinito. Tale comportamento, combinato con la relazione parabolica temperatura-frequenza del quarzo, si traduce in un disturbo che può aumentare costantemente per diversi periodi di sincronizzazione dal momento in cui viene applicato lo scalino termico, mantenendo comunque una forma a rampa su diversi periodi di sincronizzazione. Poiché è consigliabile avere errore nullo prima che la temperatura del quarzo si stabilizzi, la funzione di sensitività del controllo deve contenere due zeri per poter compensare la rampa, come risultato si può scrivere:

$$\frac{E_i(z)}{D_i(z)} = \frac{(z-1)^2 N(z)}{D(z)} \quad (3.2.19)$$

dove  $N(z)$  e  $D(z)$  sono due polinomi coprimi e  $D(z)$  è il polinomio di Schur per l'asintotica stabilità, quindi  $D(1) \neq 0$ . Omettendo banali calcoli l'espressione del controllore generale è:

$$R_i(z) = \frac{D(z) - (z-1)^3 N(z)}{(z-1)^2 N(z)}, \quad (3.2.20)$$

dove  $N(z)$  e  $D(z)$  devono essere scelti in modo tale che il grado complessivo della 3.2.20 sia non negativo per ragioni di realizzabilità. Pertanto un semplice ed efficace controllore nella forma 3.2.20 lo si ottiene scegliendo  $D(z) = (z-\alpha)^3$  e  $N(z) = 1$ , ossia:

$$R_i(z) = \frac{3(1-\alpha)z^2 - 3(1-\alpha)z + 1 - \alpha^3}{(z-1)^2} \quad (3.2.21)$$

che produce:

$$\frac{E_i(z)}{D_i(z)} = \frac{(z-1)^2}{(z-\alpha)^3} \quad (3.2.22)$$

Il controllore 3.2.21 fa convergere l'errore esponenzialmente a zero per disturbi ad impulso, scalino e rampa se il parametro  $\alpha$  assume un valore compreso tra (0,1). Valori prossimi a zero portano l'errore a rapida convergenza anche se con qualche residuo di errore dovuto al jitter, mentre valori prossimi a uno riducono il jitter a discapito di un errore che converge più lentamente. È stato trovato 3/8 essere un buon valore di default. Si ricorda che ogni  $\alpha$  in (0,1) conserva la stabilità, per cui una regolazione fine in funzione delle prestazioni può essere operata se lo

si desidera. Inoltre la natura decentralizzata del controllo permette di mescolare nodi con  $\alpha$  diversi senza soluzione di continuità. La generalità della 3.2.15 resta valida indipendentemente dal dispositivo considerato.

La nuova funzione di trasferimento che lega ingresso ed uscita del sistema diventa:

$$\frac{U_i(z)}{E_i(z)} = -R_i(z) \quad (3.2.23)$$

e dalla trasformata inversa di  $\mathcal{Z}$  si ottiene la seguente semplice legge di controllo nel dominio del tempo discreto:

$$u_i(k) = 2u_i(k-1) - u_i(k-2) \quad (3.2.24)$$

$$-3(1-\alpha)e_i(k) + 3(1-\alpha^2)e_i(k-1) \quad (3.2.25)$$

$$-(1-\alpha^3)e_i(k-2). \quad (3.2.26)$$

Un ultimo punto da esaminare è come ridurre il tempo di convergenza dell'errore all'avvio del nodo. A tal fine, la prima fase è eseguita utilizzando il *controllore deadbeat* ottenuto riscrivendo  $E_i(z)/D_i(z) = (z-1)/z^2$ . Questo controllore potrebbe non rigettare un disturbo a rampa ed è molto sensibile ai jitter, tuttavia garantisce la convergenza a zero dell'errore in due passi. Sarebbe quindi poco adatto per il normale funzionamento ma è molto utile per una rapida inizializzazione. Poi, nella seconda fase lo stato del controllore 3.2.20 viene inizializzato con quello della prima fase per la continuità e il controllore 3.2.21 prende il sopravvento.

### 3.2.4 Criticità dei disturbi di origine termica

I nodi WSN sono tipicamente dispositivi compatti in cui questioni legate al power obbligano a ridurre l'hardware all'essenziale. Come tale, se i controlli ideati mirano a una distribuzione su larga scala, è ragionevole supporre che l'unico sensore di temperatura disponibile sia sul processore del nodo. Detto processore è montato su un circuito normalmente vicino al quarzo e all'interno di un involucro. Trascurando la diffusione termica nella circuiteria e supponendo che le correnti siano così basse da produrre un calore rilevante per effetto Joule, allora il calore fluisce principalmente dall'ambiente esterno all'involucro del nodo e poi attraverso l'aria interna si propaga sul processore e sul quarzo, più o meno in parallelo. Questo consente una descrizione preliminare dei fenomeni termici di interesse come il modello LTI<sup>1</sup>

---

<sup>1</sup>Lineare Tempo Invariante.

tempo continuo:

$$\begin{cases} C_c \frac{d\theta_c(t)}{dt} = G_{ec}(\theta_e(t) - \theta_c(t)) - G_{ca}(\theta_c(t) - \theta_a(t)) \\ C_a \frac{d\theta_a(t)}{dt} = G_{ca}(\theta_c(t) - \theta_a(t)) - G_{ap}(\theta_a(t) - \theta_p(t)) \\ \quad \quad \quad - G_{ax}(\theta_a(t) - \theta_x(t)) \\ C_p \frac{d\theta_p(t)}{dt} = G_{ap}(\theta_a(t) - \theta_p(t)) \\ C_x \frac{d\theta_x(t)}{dt} = G_{ax}(\theta_a(t) - \theta_x(t)) \end{cases} \quad (3.2.27)$$

dove  $C_{c,a,p,x}$  e  $\theta_{c,a,p,x}$  sono rispettivamente le capacità termiche e le temperature di involucro, aria interna, processore e quarzo,  $\theta_e(t)$  è la temperatura (esogena) esterna, e  $G_{jk}$  la conduttanza di temperatura tra i due elementi denotati dal pedice  $j, k \in \{e, c, a, p, x\}$ , con la convenzione appena introdotta.

A fronte di una campagna di simulazioni con il modello (3.2.27) si riportano i risultati ottenuti e la metodologia seguita per ottenerli. Per stabilire un transitorio di riferimento e valutare la reiezione di un disturbo termico, si è ipotizzato che la temperatura esterna variasse in un arco di tempo  $T$  di una quantità massima  $\Delta\theta_e$ , partendo da una condizione in cui il sistema fosse in equilibrio. Poi è stato alimentato l'ingresso del modello (3.2.27) con differenti valori iniziali di temperatura esterna analizzando il conseguente comportamento attraverso la misura della temperatura  $\theta_p$  e  $\theta_x$  da compensare. E' stato piuttosto facile osservare (e il lettore lo può facilmente verificare), sull'orizzonte temporale di alcuni periodi di sincronizzazione, che cambiando leggermente qualche conduttanza o capacità (dove per "leggermente" si intende di una quantità compatibile con la variabilità del processo produttivo dei componenti hardware di un nodo o ad esempio dovuta agli effetti di condizioni impreviste come l'umidità), l'effetto può essere talmente rilevante da rendere inutile se non addirittura dannoso il processo di compensazione in anello aperto. Naturalmente il problema è meno rilevante su lassi di tempo più ampi, ma date le condizioni tipicamente imposte sul controllo della sincronizzazione in retroazione, o il controllo in anello aperto agisce in pochi periodi al massimo oppure è del tutto inutile per usi pratici.

Per supportare ulteriormente l'idea di cui sopra, si consideri come esempio rappresentativo il metodo di compensazione della temperatura proposto in [23]. L'idea principale di questo lavoro è di realizzare una sincronizzazione compensata in temperatura attraverso la costruzione progressiva del modello temperatura-frequenza del quarzo ottenuto dal campionamento della temperatura del nodo. E' abbastanza intuitivo comprendere che tale meccanismo è adatto per sistemi di sincronizzazione/compensazione basati ad esempio sulla regressione, mirando così ad un periodo di sincronizzazione intrinsecamente più lento di quello quantificato per la finestra di regressione che si estende su diversi periodi. Tuttavia, lo stesso metodo di com-



pensazione non potrebbe agire se si utilizza un periodo di sincronizzazione pari o addirittura minore a quello usato per campionare la temperatura, da qui si intuisce la forza della retroazione di FLOPSYNC.

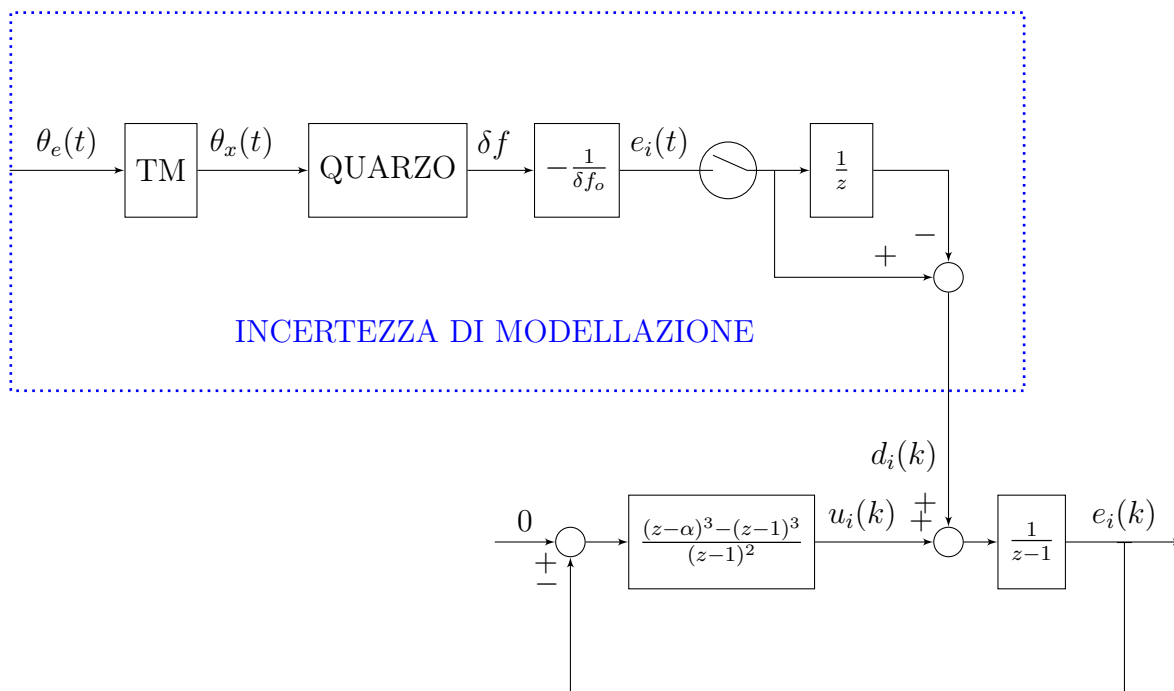


Figura 3.2.3: Azione della componente termica del disturbo sul controllo. *TM* indica il modello termico di equazioni 3.2.27, mentre il blocco *QUARZO* rappresenta la caratteristica temperatura-frequenza del quarzo.

Si ribadisce ancora una volta che nel disturbo sono contenute tutte le cause di variazione del clock. Il contributo introdotto dall'offset è costante pertanto l'azione integrale del controllo lo elimina totalmente, quello legato alla componente di jitter agisce in una banda di frequenza piuttosto alta e perciò non si può fare altro che limitare la sensibilità del controllo nella sua banda d'azione compatibilmente con gli altri vincoli del controllo, resta infine quello termico, unico vero disturbo da rigettare. Pertanto, lo schema di figura 3.2.3 mette in risalto l'azione del disturbo della sola componente termica sul sistema. Per come è stato ricavato il controllore, possiamo affermare con fermezza che indipendentemente dal modello termico piuttosto che dall'aderenza alla realtà della caratteristica temperatura-frequenza del quarzo specificata nei datasheet dal costruttore, tutta l'incertezza del modello ricade nella modellazione del disturbo. Perciò il sistema controllato può essere considerato privo di qualsiasi forma d'incertezza.

Riassumendo, quando gli schemi di retroazione adottano un approccio stile FLOPSYNC e vi sono una vasta gamma di variazioni termiche, l'unico approccio ragionevole da un punto di vista ingegneristico è quello di fornire indicazioni per

la selezione sia di  $\alpha$  che di  $T$ , vincolando il valore di picco e il tempo di recovery dell'errore di sincronizzazione a fronte di un evento termico, sulla base delle informazioni relative al range di temperatura di esercizio e la velocità massima di variazione della temperatura. Questo può essere utilizzato per configurare il nodo offline a partire dai dati nominali, come illustrato nel seguito.

### 3.2.5 Limitazione dell'errore e taratura del controllore

Questa sezione condurrà alla scelta dei parametri del periodo di sincronizzazione  $T$  e all'unico parametro del controllore relativo alla convergenza  $\alpha$ , basandoci esclusivamente sulle informazioni nominali dell'oscillatore al quarzo, condizioni operative e limiti di errore (per semplicità ometteremo il pedice  $i$  in quanto la medesima analisi è applicabile ad ogni nodo della WSN), ovvero:

- la temperatura del quarzo per la frequenza nominale  $\theta_o$  e il coefficiente di temperatura del quarzo  $\beta$ ,
- temperatura esterna minima  $\theta_{e,min}$  e quella massima  $\theta_{e,max}$  di esercizio del nodo, quali ad esempio per un nodo outdoor potrebbero essere la massima temperatura in estate e la minima in inverno,
- la massima variazione  $r_{\theta,max}$  di temperatura considerata rispetto al tempo,
- la massima ampiezza  $\Delta\theta_{max}$  di temperatura in cui può oscillare un singolo evento termico, in quanto è molto improbabile che detta oscillazione non abbracci l'intera gamma  $(\theta_{e,min}, \theta_{e,max})$ ,
- la massima ampiezza  $e_{max}$  per il valore di picco tollerabile dell'errore controllato (cioè, quello misurato all'istante di sincronizzazione) dopo un evento di temperatura,
- il tempo massimo  $t_{r,max}$  (un multiplo di  $T$  data la regolazione assunta) dopo un evento in cui l'ampiezza dell'errore deve ritornare al di sotto dei limiti imposti  $\bar{e} < e_{max}$ .

Naturalmente lo scopo è ideare limiti conservativi, concepito però in modo tale che essi possono essere ottenuti in modo semplice, a vantaggio della rapidità di configurazione del sistema e interpretabilità da parte dell'utente.

Per iniziare l'analisi, si supponga che inizialmente il nodo sia in equilibrio termico alla temperatura esterna  $\theta_{e,min}$  e che poi la sua temperatura  $\theta$  salga fino a  $\theta_{e,min} + \nabla\theta$  con un transitorio esponenziale caratterizzato da un rate massimo  $r_{\theta,max}$ , cioè:

$$\theta(t) = \theta_{e,min} + \Delta\theta_{max}(1 - \exp^{-t \frac{r_{\theta,max}}{\Delta\theta_{max}}}). \quad (3.2.28)$$

Questo *stimolo*, insieme a quello generato dal passaggio da  $\theta_{e,max} - \Delta\theta_{max}$  a  $\theta_{e,max}$  e quelli con partenza e arrivo invertiti (la necessità per tutti e quattro deriva dalla posizione generalmente asimmetrica di  $\theta_{e,max}$  e  $\theta_{e,min}$  rispetto a  $\theta_o$ ) rappresentano le “peggiori” sollecitazioni possibili compatibilmente con le condizioni assunte, pur essendo abbastanza realistica da non produrre risultati troppo conservativi, in quanto potrebbero essere causati da una fase di potenza radiante (si pensi a un nodo mobile entrare o uscire da un tunnel, per esempio). Supponendo, per spostare ulteriormente l’analisi verso il caso pessimo, che l’aumento di temperatura inizi immediatamente dopo un istante di sincronizzazione, corrispondente a  $k = 0$  per lo stesso motivo di cui sopra, e che la temperatura del quarzo segue istantaneamente quella esterna, si avrebbe:

$$d^s(k) = -\frac{\beta}{10^6} \int_{kT}^{(k+1)T} (\theta(t) - \theta_o)^2 dt. \quad (3.2.29)$$

Una volta note le caratteristiche del quarzo ( $\beta$  e  $\theta_o$ ) e le condizioni operative ( $\theta_{e,max}$ ,  $\theta_{e,min}$ ,  $\Delta\theta$  e  $r_{\theta,max}$ ) quattro applicazioni della 3.2.29 per i suddetti casi pessimi di cambiamenti di temperatura permettono di calcolare il caso pessimo del disturbo  $d_{1,2,3,4}^s(k)$  per un singolo valore di  $T$ . Successivamente, alimentando il sistema dinamico 3.2.22 con  $d_{1,2,3,4}^s(k)$ , inizializzato ad un conveniente equilibrio, prontamente fornirà la massima ampiezza di errore e il numero di passi necessari a recuperare  $\bar{e}$  per un dato valore di  $\alpha$ .

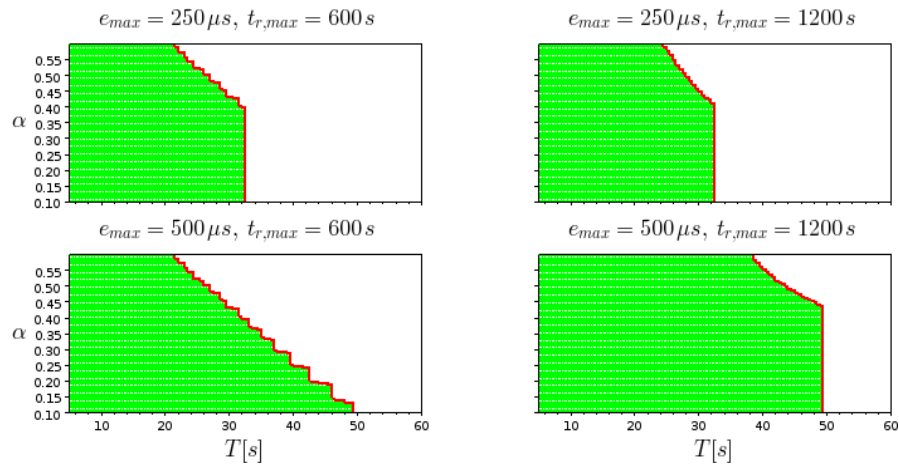
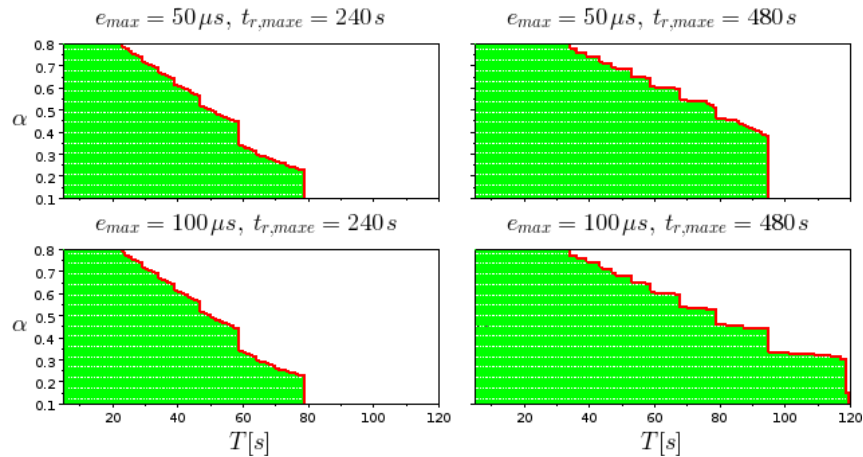
Riassumendo, con un ragionevole sforzo computazionale (offline), si possono ottenere da  $\beta$ ,  $\theta_o$ ,  $\theta_{e,max}$ ,  $\theta_{e,min}$ ,  $\Delta\theta$  e  $r_{\theta,max}$ , l’errore di picco  $e_{max}$  e il tempo di recovery in passi  $t_{r,max}$  per  $T$  ed  $\alpha$ . Due esempi sono riportati in figura 3.2.4 e 3.2.5, denominati caso 1, il quale assume un ambiente operativo piuttosto “severo”, e caso 2 che si riferisce ad una situazione più “dolce” (ad esempio a un nodo indoor) ma presenta un quarzo con una dipendenza maggiore dalla temperatura (ricordiamo che i datasheet forniscono un valore che rappresenta il caso pessimo) le cui specifiche di progetto sono riportate in tabella 3.1 .

Si supponga ora che le specifiche richiedano un certo  $e_{max}$  e  $t_{r,max}$  per il recovery di un dato  $\bar{e}$ , fissato a  $20\mu s$  per entrambi i casi. I risultati delle figure 3.2.4 e 3.2.5 possono facilmente essere usati per determinare la coppia  $(T, \alpha)$  che rende attuabile il progetto come indicato dalla regione verde delle figure 3.2.6 e 3.2.7 (l’effetto quantizzato che si osserva nelle figure è dovuto alla granularità del periodo di sincronizzazione). Coerentemente con la natura dei casi 1 e 2, in quest’ultimo

	caso 1	caso 2	
$\beta$	0.025	0.04	$ppm/^\circ C^2$
$\theta_0$	25	25	$^\circ C$
$\theta_{e,min}$	-20	15	$^\circ C$
$\theta_{e,max}$	50	22	$^\circ C$
$\Delta\theta_{max}$	25	5	$^\circ C$
$r_{\theta_{max}}$	8	0.5	$^\circ C/min$

Tabella 3.1: Specifiche di progetto per il caso 1 ed il caso 2.

requisiti più severi sono stati formulati, per dimostrare l'efficacia e la flessibilità della metodologia proposta.

Figura 3.2.6: Coppie  $(T, \alpha)$  per il caso 1 che rendono il progetto attuabile.Figura 3.2.7: Coppie  $(T, \alpha)$  per il caso 2 che rendono attuabile il progetto.

Visti i risultati esemplificati nelle figure 3.2.6 e 3.2.7, come ulteriore ausilio per finalizzare la selezione di una coppia  $(T, \alpha)$  si può considerare l'ampiezza ad alta

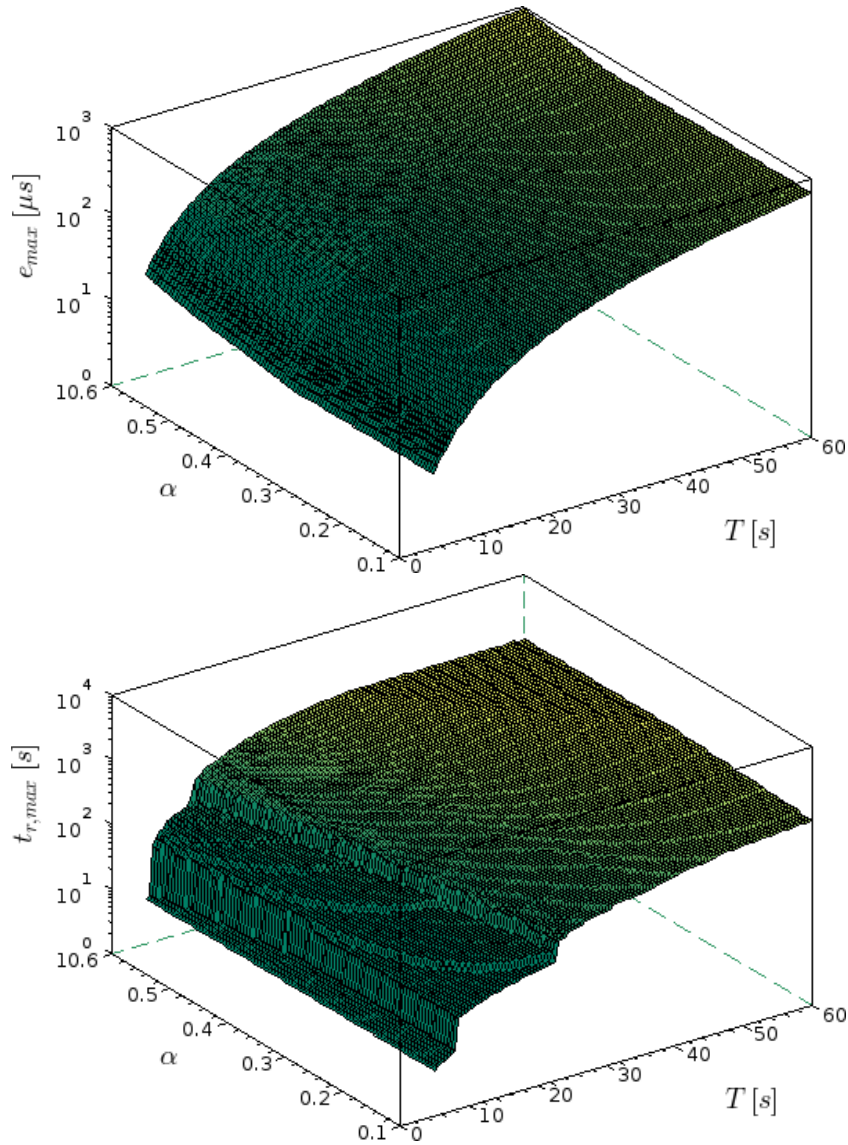


Figura 3.2.4: Caso 1: errore di picco (sopra) e tempo di recovery (sotto) come funzione di  $T$  ed  $\alpha$ .

frequenza della risposta in frequenza della 3.2.22, cioè, con la notazione di questa sezione:

$$\left| \frac{E(e^{j\vartheta})}{D(e^{j\vartheta})} \right|_{\vartheta=\pi} = \frac{4}{(1+\alpha)^3} \quad (3.2.30)$$

e osservare che valori più bassi di  $\alpha$  permettono grandi periodi di sincronizzazione a costo di alcune amplificazioni del jitter ad alta frequenza, proprio come anticipato nella sezione di sintesi del controllore. Non avendo trattato finora l'aspetto del consumo di potenza, si consideri vera a priori (fino alla seguente e dettagliata trattazione) quanto segue: la curva rossa al bordo della regione ammissibile delle figure 3.2.6 e 3.2.7 può essere vista come una sorta di curva di Pareto per la qualità

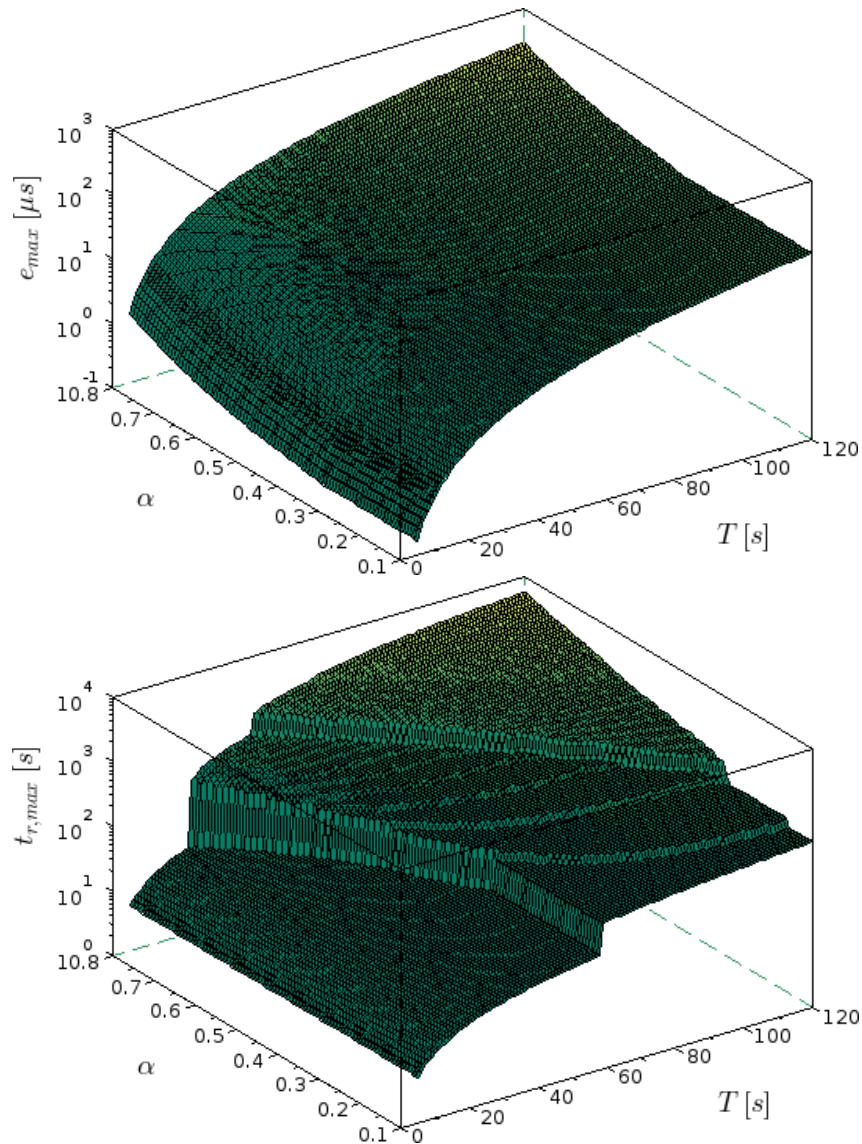


Figura 3.2.5: Caso 2: errore di picco (sopra) e tempo di recovery (sotto) come funzione di  $T$  ed  $\alpha$ .

della sincronizzazione (alto  $\alpha$ , basso jitter) contro il consumo di energia (grande periodo di sincronizzazione  $T$ ).

### 3.3 Dal problema di controllo allo schema di sincronizzazione

Nella sezione 3.1 è stato anticipato che la soluzione al problema di sincronizzazione, per il modo in cui è stata modellata, porta ad un meccanismo unico che combina sia la sincronizzazione del clock che la compensazione dello skew. Questa sezione

apre la strada alla descrizione dello schema proposto, analizzando i risultati della sezione 3.2 in tale prospettiva.

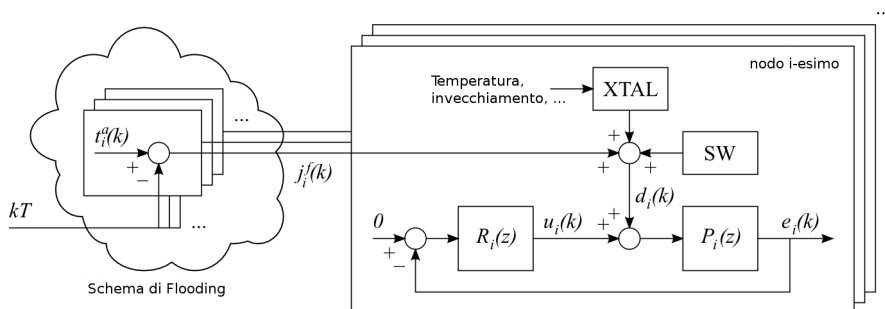


Figura 3.3.1: Struttura del controllo decentralizzato;  $j_i^f(k)$  rappresenta il jitter che ha origine dallo schema di flooding, mentre  $XTAL$  e  $SW$  rappresentano quelli introdotti dal quarzo e dal software.

In primo luogo, si ricorda nuovamente l'assunzione della presenza di uno schema di flooding che bypassa interamente il protocollo e null'altro (come mostra la figura 3.3.1) in modo da isolare il fenomeno fisico da controllare (ossia le fluttuazioni del clock locale) e posizionare le parti di monitoraggio e attuazione esattamente ai limiti di detto fenomeno (si veda a tale scopo la discussione in [24] per apprezzare quanto questo sia importante in numerosi problemi di controllo relativi all'informatica). Modellare questo fenomeno nel modo appena mostrato è piuttosto semplice e permette di lasciare al controllore ed alla retroazione il compito di rigettare i disturbi che introducono fluttuazioni di frequenza, disinteressandosi della loro origine. La soluzione quindi non richiede alcuna misurazione della temperatura, né ipotesi sul modello temperatura-frequenza del quarzo da controllare.

In secondo luogo, il controllo proposto è totalmente decentralizzato. Interazioni tra nodi non-reference sono richieste solo per il flooding e non hanno nulla a che fare con il calcolo dei segnali di controllo. Ciò, oltre a favorire un basso overhead di traffico, implica formalmente che la proprietà di convergenza dell'errore (cioè stabilità) sia garantita a livello di nodo, di conseguenza per l'intera rete. A questo si aggiunge la possibilità di sincronizzare nodi con quarzi differenti oltre che con hardware diverso.

In terzo luogo, i controllori della sezione 3.2 non usano *timestamp*. Questo è un passo in avanti rispetto allo stato dell'arte in quanto l'informazione richiesta per la sincronizzazione si sposta dal contenuto del bsp al *tempo di arrivo* del bsp. Questa innovazione porta con sé anche l'eliminazione di alcune incertezze dei segmenti che compongono il ritardo di trasmissione. Si ricorda nuovamente che il ritardo di trasmissione può essere solo stimato. Pertanto eliminare fattori che creano incertezza come la generazione di timestamp (pur se fatta a livello hard-

ware) e l'incapsulamento nel pacchetto a livello MAC migliorano indubbiamente l'accuratezza di sincronizzazione. Inoltre l'assenza di timestamp riduce la quantità di dati trasmessi col bsp e pertanto riduce il consumo di potenza dovuto alla trasmissione/ricezione. Si tenga presente che meno dati vengono trasmessi e meno tempo si dovrà tener accesa la radio sia in trasmissione che in ricezione.

In quarto luogo, ogni singolo controllore può essere tarato *offline* (prima della sua messa in campo), per garantire un determinato livello di prestazioni entro il quale lo schema dovrà rimanere. Questo può essere fatto semplicemente attraverso la conoscenza di pochi parametri nominali del quarzo (riportati sul datasheet dal costruttore) e le condizioni operative in cui il nodo si troverà ad operare (esempio, su di un ghiacciaio, all'interno di un edificio ecc...).

Infine, e la più importante per quanto segue, i controllori proposti non modificano il clock hardware del nodo. Infatti la modifica del tempo di attesa prima del prossimo evento di sincronizzazione corrisponde alla  $u_i$  del segnale di controllo che può essere implicitamente visto come la quantità di correzione da apportare al clock locale, rispettando sostanzialmente il punto di vista classico sincronizzazione/compensazione (cfr. sezione 3.1). Tuttavia, il modo in cui il controllo viene calcolato produce intrinsecamente sincronizzazione del clock e compensazione dello skew *congiuntamente*, portando alla necessità di eliminare l'offset iniziale come discusso di seguito, mentre il modo in cui l'azione di controllo viene azionata assicura strutturalmente la monotonicità del tempo locale.

Si osservi innanzi tutto che  $e_i(k)$  2.2.2, vedi anche figura 3.2.1, corrisponde a  $e_i(t(k)^-)$  secondo la 3.2.5, per cui un segnale di errore cattura tutte le cause di diversità del clock, compresi gli effetti skew/drift tra una sincronizzazione e l'altra. Detto questo, si supponga che, tanto per fare un esempio per illustrare l'idea di cui sopra, si voglia utilizzare il sistema di controllo di figura 3.2.2 per ottenere solo la sincronizzazione dell'orologio. Adottando la notazione solita, questo potrebbe significare, assumendo:

$$t_i(t(k+1)) = t_i(t(k)) + T + u_i(k) \quad (3.3.1)$$

$$= t(k+1) \quad (3.3.2)$$

$$= t(k) + T, \quad (3.3.3)$$

che:

$$u_i(k) = t(k) - t_i(t(k)), \quad (3.3.4)$$



cioè,  $R_i(z) = 1$  diversamente da quanto trovato nella sottosezione 3.2.3. Il controllore  $R_i(z) = 1$  manca chiaramente dell'azione integrale, ragion per cui non potrà portare l'errore a zero (si veda la figura 2.6.1). Tutto questo per dire che il controllore proposto *generalizza* quello tipico esclusivo per la sola sincronizzazione del clock, in modo da comprendere la compensazione di skew/drift. Naturalmente si potrebbe realizzare lo schema di figura 3.2.2 con  $R_i(z) = 1$  (per la sola sincronizzazione) e poi chiudere un altro anello intorno ad esso per la compensazione. Questo non sarebbe scorretto, ma porterebbe solo ad una inutile complicazione dell'algoritmo di controllo considerando che ogni risultato così ricavato potrebbe essere equivalentemente ottenuto con un'opportuna  $R_i(z)$  nel sistema di figura 3.2.2.

Venendo a come il segnale di controllo viene azionato, avendo stabilito che  $e_i(k)$  è governato dall'integratore del controllore, significa raggiungere sia la sincronizzazione che la compensazione. Si noti che la regolazione del tempo di attesa, invece di modificare il clock hardware del nodo, può essere vista come la distribuzione della correzione per entrambi, sincronizzazione e compensazione (non solo per la compensazione, come per l'esempio di figura 2.6.1) per tutto il periodo di sincronizzazione, invece di applicare la relativa parte di sincronizzazione di detta correzione tutta al suo inizio. Poiché il tempo di attesa inizia dal tempo *expected* del bsp, la proprietà di monotonicità è quindi garantita per progettazione. Considerato quanto illustrato precedentemente, allora:

- la quantità  $e_i(k)$  può essere usata per calcolare *a posteriori* l'intervallo del tempo locale corrispondente a  $T$  nel tempo globale, al termine di un intervallo di sincronizzazione,
- mentre la quantità  $u_i(k)$  può essere usata per *prevedere* l'ampiezza dell'intervallo di attesa locale.

Questo è il modo in cui FLOPSYNC fornisce un clock virtuale monotono base, che segna il tempo con pacchetti di sincronizzazione e possono differire dal riferimento solo a causa di un offset e/o di jitter. Tutto ciò può essere sfruttato, se necessario, per avere diversi clock virtuali coesistenti sullo stesso nodo, utilizzando un'unica base dei tempi, per esempio sia monotono che non monotono.

### 3.3.1 Un clock virtuale monotono

Un clock virtuale ha il compito di traslare il tempo locale  $t_i$  (non corretto) in quello di globale  $t$ . Le informazioni utilizzate sono il tempo di  $t_i$ , il valore di  $t(k)$  quando l'ultimo pacchetto di sincronizzazione è stato inviato, il tempo locale  $t_i^e(k)$

quando l'arrivo del pacchetto è stato previsto e la correzione del clock  $u_i(k)$ . La traslazione del tempo avviene riscrivendo  $t_i$  come  $t_i^e(k) + (t_i - t_i^e(k))$ . Il primo termine è convertito nel tempo di riferimento sostituendo semplicemente  $t_i^e(k)$  con  $t(k)$  poiché si tratta dell'istante di tempo in cui il nodo reference ha inviato l'ultimo pacchetto di sincronizzazione. Il termine  $t_i - t_i^e(k)$  viene poi corretto con una interpolazione lineare impiegando  $u_i(k)$  per la stima dello skew. La risultante formula di traslazione pertanto è la seguente

$$t = t(k) + (t_i - t_i^e(k)) \frac{T}{T + u_i(k)} \quad (3.3.5)$$

La figura 3.3.2 riporta un esempio schematizzato di quanto detto che potrà chiarire ulteriormente il ragionamento seguito.

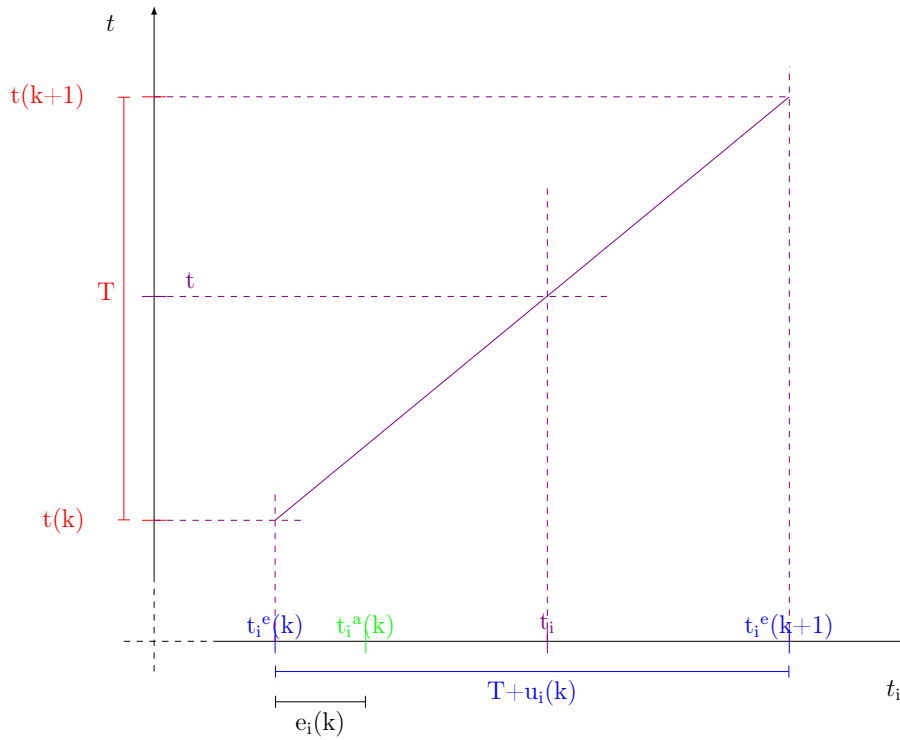


Figura 3.3.2: Traslazione tra tempo locale e tempo globale con clock virtuale monotono.

Si può discutere sulla scelta di  $t_i^e(k)$  invece di  $t_i^a(k)$ , che è apparentemente una migliore stima del tempo locale corrispondente a  $t(k)$ , ma questo è il **punto chiave per realizzare un clock monotono**, poiché trascurando la misura più accurata di  $t_i^a(k)$  possiamo affermare che  $e_i(t(k)^+) = e_i(t(k-1)^-)$ , mentre ci si affida alla compensazione prodotta dalla rete di retroazione per minimizzare la differenza tra il tempo di arrivo previsto e quello effettivo. L'equazione 3.3.5 viene chiamata formula di traslazione del tempo in *avanti*, che traduce dal tempo locale a quello

globale di riferimento. In alcuni casi, ad esempio quando si schedulano task espressi nel tempo globale, può essere utile eseguire una traslazione del tempo all'*indietro*, cioè, dal tempo globale a quello locale. Ciò richiede semplicemente di invertire la 3.3.5, il cui risultato è:

$$t_i = t_i^e(k) + (t - t(k)) \frac{T + u_i(k)}{T}. \quad (3.3.6)$$

## 3.4 Il protocollo di sincronizzazione FLOPSYNC

Questa sezione mostra per intero il protocollo FLOPSYNC. Di seguito verranno rimosse tutte le assunzioni fatte fino ad ora e saranno presentate le quattro componenti che compongono lo schema di sincronizzazione FLOPSYNC: il *controllore*, lo *schema di flooding*, il *clock virtuale* e lo *schema di re-sincronizzazione*. Le prime due componenti vengono utilizzate dal protocollo ogni periodo  $T$  e si occupano rispettivamente di inoltrare il bsp all'intera rete e generare il segnale di sincronismo. Il clock virtuale invece viene chiamato ogni qual volta il sistema operativo del nodo WSN necessita del tempo globale. Infine, lo schema di re-sincronizzazione viene eseguito al boot del nodo oppure quando la sincronizzazione viene persa, per esempio quando a causa di interferenze radio non viene ricevuto il bsp.

Il protocollo FLOPSYNC opera a livello MAC dello stack radio, come si può vedere dalla figura 3.4.1, ed è completamente trasparente ai livelli successivi che possono quindi operare come se non ci fosse.

Prima di passare alla descrizione delle componenti di FLOPSYNC è bene integrare il modello proposto in figura 3.2.1, volto esclusivamente a raggiungere gli obiettivi di accuratezza di sincronizzazione e bassa complessità computazionale, con le informazioni necessarie per avere un schema ad alta efficienza energetica. A tale scopo si guardi il nuovo modello proposto in figura 3.4.2. Per raggiungere un'alta efficienza energetica FLOPSYNC sfrutta il concetto di finestra di ricezione. La finestra di ricezione indicata in figura 3.4.2 con  $w$  attorno a  $t_i^e(k)$ , è il tempo entro il quale il nodo attende la ricezione del bsp. Per cui il nodo si porterà nello *stato attivo* (se non lo era già) solo all'istante  $t_i^e(k) - w$  e ci rimarrà al più fino all'istante  $t_i^e(k) + w$  nel caso in cui per qualche ragione non riceva il bsp, altrimenti non appena terminata la ricezione del bsp il nodo tornerà nello *stato low-power*, ammesso che non abbia altre operazioni da svolgere. Di seguito verrà chiarito come scegliere  $w$  per non avere una alta perdita di pacchetti.

### Modello ISO dello stack adattato a ZigBee

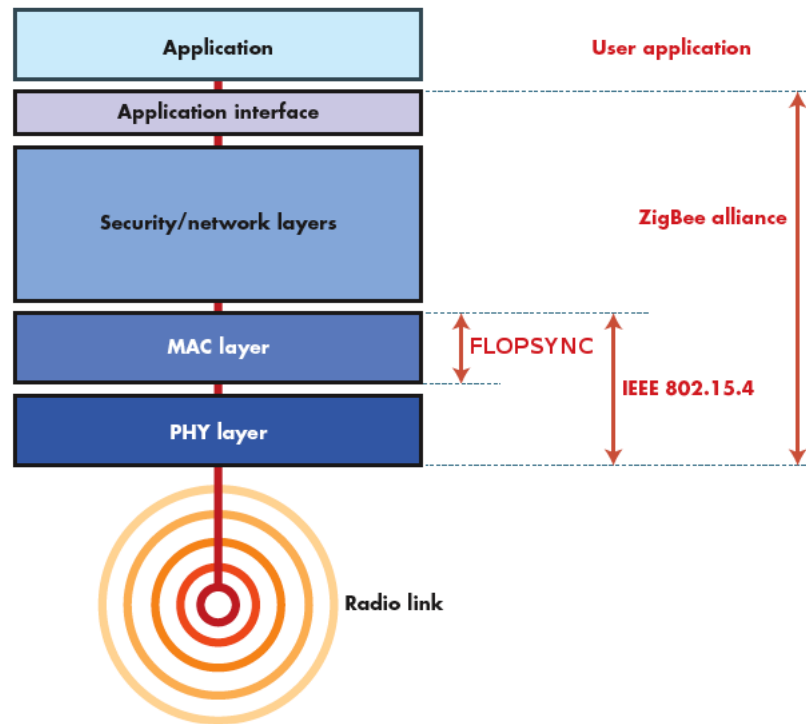


Figura 3.4.1: Collocazione di FLOPSYNC all'interno dello stack radio ZigBee.

#### 3.4.1 Schema di flooding

Lo scopo dello schema di flooding è quello di ricevere e ritrasmettere il bsp, e derivare la misura di  $t_i^a(k)$  opportunamente corretta con il tempo di trasmissione (può essere facilmente ricavato dalla conoscenza del data rate della radio e dal numero di hop).

Lo schema di flooding utilizzato dal protocollo FLOPSYNC è quello proposto da *F. Ferrari et al. nel 2011* [25]: glossy. Glossy sfrutta le interferenze costruttive di simbolo per lo standard IEEE 802.15.4 per inondare la rete velocemente e trasmettere implicitamente la sincronizzazione del tempo. Senza entrare nella trattazione matematica che c'è dietro questo schema, proviamo a spiegare brevemente l'importante risultato. Per interferenze costruttive di simbolo si intende che se due o più trasmissioni concorrenti del medesimo pacchetto vengono intraprese ad una distanza temporale al di sotto del mezzo microsecondo, i ricevitori (uno o più) riceveranno correttamente il contenuto inviato con una probabilità del 99.9%. Questo è il vero punto innovativo di questo meccanismo di flooding che lo rende veloce e versatile.

Il protocollo FLOPSYNC, in accordo con glossy, richiede che il canale radio

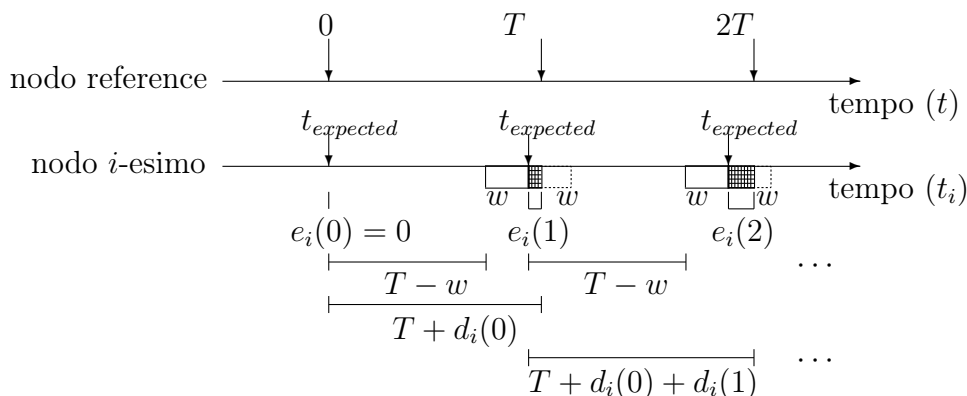


Figura 3.4.2: Modello del sistema controllato con finestra.

sia privo di contese di accesso quando i pacchetti bsp sono inoltrati nella rete in modo da non alterare l'informazione di sincronizzazione intrinsecamente contenuta nell'invio/ricezione del pacchetto. Quindi, se si usa un periodo di sincronizzazione  $T$ , per una breve finestra di tempo, lo schema di flooding prende il controllo del livello MAC di tutti i nodi della WSN.

Il tempo della suddetta finestra dipende dal tempo richiesto per "inondare" la rete con i bsp, ma in generale è inferiore a qualche millisecondo. In realtà la finestra di tempo (ossia quella indicata in precedenza con la lettera  $w$ ) per il nostro schema è sensibilmente inferiore al millisecondo ed è dovuto alla capacità di  $w$  di "adattarsi" in tempo reale. Questo accoppiato con la possibilità di poter scegliere larghi periodi di sincronizzazione (60s è un valore comune) e con la qualità di sincronizzazione fornita con la compensazione di skew/drift, porta ad un impatto davvero minimo dell'utilizzo di banda. È importante sottolineare che FLOPSYNC, eccetto che per la finestra di tempo di sincronizzazione, non impone alcun vincolo sull'utilizzo dello stack radio: questo significa che un nodo può utilizzare un comune MAC low-power listening, ad esempio.

Inoltre FLOPSYNC non richiede nessun contenuto particolare per il pacchetto di sincronizzazione, il che apre un'altra opportunità di ottimizzazione del consumo energetico. Il bsp può essere davvero piccolo in termini di byte, riducendo l'overhead del consumo di potenza sia lato trasmettitore che lato ricevitore. Ad esempio lo schema di flooding proposto richiede solamente un campo di un byte per il bsp usato come contatore di ritrasmissione. Questo consente di inondare la rete con un pacchetto di solo un byte di payload. Ad oggi, nessun altro schema lavora con un payload così piccolo. Inoltre lo schema di flooding permette di fare

piggybacking per inviare comandi all'intera rete con lo stesso periodo usato per la sincronizzazione. Pertanto la lunghezza del bsp è variabile e si riduce ad un solo byte quando non ci sono comandi da trasmettere.

### 3.4.2 Controllore

Lo scopo del controllore FLOPSYNC dovrebbe essere ormai abbastanza chiaro al lettore. Il suo compito è quello di calcolare la  $u_i(k)$  e di conseguenza  $t_i^e(k+1)$ , basandosi su  $t_i^a(k)$ . La conoscenza di  $t_i^e(k+1)$  permette di schedare quando lo schema di flooding dovrà prendere il controllo della radio, mentre  $u_i(k)$  è usato dal o dai clock virtuali.

#### Computazione del segnale di controllo

La computazione è fatta per mezzo dell'equazione 3.2.24. Il controllore necessita di sole quattro locazioni di memoria per memorizzare i precedenti valori del controllo e dell'errore, e richiede un tempo di calcolo davvero irrisorio, inoltre può essere implementata efficientemente in aritmetica intera come verrà mostrato nel prossimo capitolo.

#### Adattamento della finestra di ricezione

FLOPSYNC adatta la finestra di ricezione  $w$  basandosi sulla varianza dell'errore di sincronizzazione. Questo è un punto cruciale sia per quanto riguarda il consumo energetico, sia per l'overhead introdotto dallo schema di flooding. Per la prima volta rispetto agli schemi di sincronizzazione esistenti (lo si può notare anche in quelli presentati nel capitolo precedente) accuratezza e efficienza energetica non sono in contrapposizione, anzi quanto più lo schema di controllo sarà in grado di predire accuratamente l'istante di trasmissione del prossimo bsp, minore sarà il tempo speso in ricezione e di conseguenza minore sarà il consumo energetico.

Il tempo  $w$  è critico in termini di power/affidabilità perché piccoli valori di  $w$  risultano in un alto rischio di perdere il pacchetto di sincronizzazione, mentre per valori alti si ha una perdita in termini di power. La scelta di  $w$  pertanto è cruciale.

Assumendo che l'errore di sincronizzazione ha una distribuzione normale standard (è ragionevole pensare che sia così), si può decidere di adattare la finestra di ricezione considerando la deviazione standard degli ultimi  $N$  errori di sincronizzazione. Sapendo che per una distribuzione normale si ha la probabilità del 99.7% che l'errore di sincronizzazione ricada entro 3 volte lo scarto quadratico medio, la scelta di  $w$  è presto fatta.

Per limitare l'uso di risorse il calcolo della deviazione standard viene fatto su blocchi di  $N$  campioni e tipicamente 8 o 16 si rivelano essere una buona quantità di valori. Considerando che il periodo di sincronizzazione  $T$  è di 60 secondi, si ha un adattamento della finestra ogni 8 o 16 minuti.

Non appena il pacchetto viene completamente ricevuto non è necessario attendere per la restante parte di  $w$ , per cui la radio può essere spenta e lo schema di flooding può provvedere al rinvio in broadcast del bsp.

Nel caso in cui il pacchetto non viene ricevuto all'interno di  $2w + packetTime$ , dove  $packetTime$  è il tempo per trasmettere il pacchetto, il bsp viene considerato perso. In questo caso il valore della finestra di ricezione viene raddoppiato. Vi possono essere al più  $n$  perdite consecutive (solitamente la soglia  $n$  è impostata a 3) di un bsp, dopo di che la sincronizzazione viene dichiarata persa e il protocollo cede il controllo allo schema di re-sincronizzazione. Sono previste anche una soglia massima e minima impostabili per  $w$ .

### 3.4.3 Clock virtuale

Lo scopo dell'orologio virtuale è quello di fornire una stima affidabile del tempo di riferimento  $t$ , compensando skew e drift attraverso le informazioni fornite dal controllore FLOPSYNC. Questa è la componente più semplice da implementare del protocollo FLOPSYNC ma anche la più utile (si rammenta che l'intero scopo della sincronizzazione è quello di fornire il tempo di riferimento ad ogni nodo della rete). L'implementazione per fornire la traslazione in avanti, ossia da tempo locale a tempo globale segue la 3.3.5. La traslazione all'indietro è invece governata dalla 3.3.6.

### 3.4.4 Schema di re-sincronizzazione

Questo componente si occupa del recupero delle sincronizzazioni perse, nonché della ricezione del primo pacchetto di sincronizzazione all'avvio di un nodo. Il suo compito è di resettare le variabili di stato del controllore FLOPSYNC, inizializzare la finestra di ricezione  $w$  al suo valore massimo ed attendere il pacchetto di sincronizzazione.

Lo stesso componente si occupa anche dell'eliminazione dell'offset. Questo avviene utilizzando un protocollo request/response di timestamp, approfittando del protocollo MAC ordinario utilizzato dai nodi quando non è in esecuzione lo schema di flooding, cioè, trasmettendo *sporadicamente* il timestamp. Per evitare incertezze sul tempo causate da un accesso a contesa, nel protocollo request/response il

pacchetto di risposta non contiene l'ora corrente ma il tempo di trasmissione dell'ultimo pacchetto di sincronizzazione inviato. Poiché questo si verifica con un periodo costante  $T$ , conosciuto da tutti i nodi, il tempo di riferimento corrispondente a tutti i futuri pacchetti di sincronizzazione può essere determinato.



# Capitolo 4

## Implementazione

Questo capitolo mostra come FLOPSYNC è stato implementato su una reale piattaforma hardware. Prima di presentare l'architettura di riferimento di un nodo, verrà spiegato brevemente come è stato implementato il tool di configurazione FLOPSYNC. Per non rendere noiosa e lunga la trattazione si è deciso di non entrare nel merito del codice se non in casi particolari. Pertanto non si pretenda che tutto sia spiegato dettagliatamente, piuttosto l'idea è quella di dare una visione d'insieme dell'implementazione. Qualora il lettore volesse entrare nel merito del codice, può scaricarlo da questo link <https://gitorious.org/flopsync> dove trova anche un'estensiva documentazione dello stesso.

### 4.1 Tool di configurazione per FLOPSYNC

Ricordando quanto detto nella sottosezione 3.2.5, per la taratura del controllore è stato creato un tool di configurazione che agevoli il calcolo dei parametri  $T$  ed  $\alpha$  in risposta ai vincoli imposti allo schema di sincronizzazione. All'utilizzatore viene presentata un'interfaccia simile a quella mostrata in figura 4.1.1 in cui gli viene richiesto di inserire le informazioni di targa del quarzo utilizzato, le condizioni operative del nodo e i vincoli imposti per lo schema di sincronizzazione.

Il tool di configurazione restituirà all'utente in output qualcosa di simile al grafico di figura 4.1.2, da cui si potrà facilmente selezionare una coppia  $T$ ,  $\alpha$  per la configurazione del controllore.

All'utilizzatore è lasciata la facoltà di privilegiare l'efficienza energetica scegliendo periodi lunghi di sincronizzazione piuttosto che l'accuratezza, a seconda della coppia  $T$ ,  $\alpha$  selezionata.

L'intero tool di configurazione è stato realizzato con uno script e richiede un'esecuzione offline rispetto al protocollo di sincronizzazione che può essere fatta su

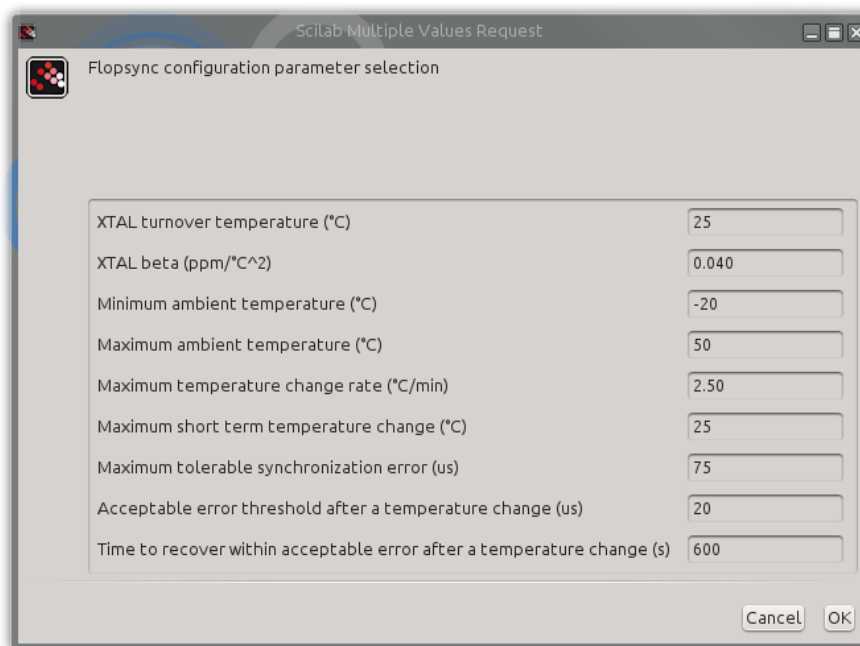


Figura 4.1.1: Tool di configurazione per il controllore.

un qualsiasi calcolatore su cui è installato il software open source Scilab<sup>1</sup>.

## 4.2 Architettura hardware-software di un mote

Quello che sino ad ora è stato chiamato nodo sensore generalmente è conosciuto anche col termine *mote*, coniato dai ricercatori dell'Università di Berkley in California nel 1998 col progetto Smartdust[26]. Attualmente possono essere reperiti sul mercato svariati tipi di mote. I più comuni sono:

- Famiglia Mica (Mica Mica2 Mica2dot MicaZ) della Crossbow Technology Inc [27]
- Famiglia Telos (Telos, Telosa, Telosb, TmoteSky ) della Moteiv Corporation [28]
- Intelmote2 della Intel [29]
- Famiglia Eyes (EyesIFX EyesIFXv1 EyesIFXv2 ) della Ember Corporation [30]

L'architettura di questi mote è molto simile, essenzialmente sono tutti l'evoluzione del nodo nominato Mica progettato presso l'Università di Berkeley. Il nodo Mica2

<sup>1</sup><http://www.scilab.org/>

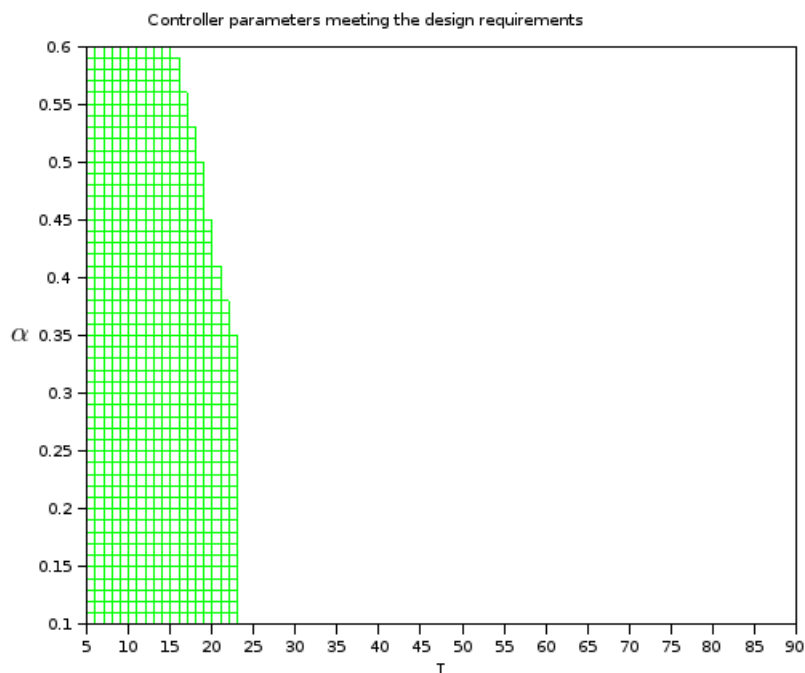


Figura 4.1.2: Output del tool di configurazione riferito alla figura 4.1.1 per la scelta  $T$ ,  $\alpha$ .

è stato per gran tempo la tecnologia di riferimento del mondo della ricerca e non, anche se attualmente è da considerarsi sorpassato dai Telosb. Le componenti di base di un mote sono le seguenti: il microcontrollore, il transceiver, la memoria esterna, la sorgente di alimentazione e uno o più sensori, come mostrato in figura 4.2.1

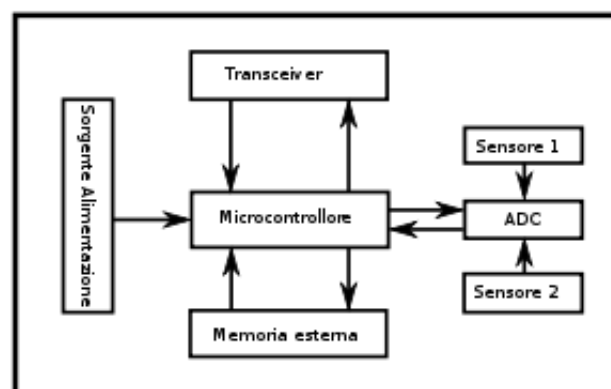


Figura 4.2.1: Architettura di un mote.

Per quanto riguarda il software anche in questo caso non vi è un unico sistema operativo ma ve ne sono diversi, tra cui i più diffusi sono TinyOS [31] sviluppato

dalla stessa Berkley e Contiky [32].

Per l'infrastruttura di comunicazione generalmente viene utilizzato l'ormai affermato ZigBee[33] basato sullo standard IEEE 802.15.4 [34] per Wireless Personal Area Networks (WPAN). Pur disponendo di un transfer rate piuttosto basso 250Ksa/s ha il vantaggio di avere consumi molto contenuti. E' questa la motivazione principale che ha reso ZigBee uno standard de facto per le applicazioni embedded e non solo.

Per testare FLOPSYNC, nessuna delle piattaforme hardware software elencate è stata utilizzata in quanto si è deciso di fare qualcosa di diverso e le motivazioni sono di seguito riportate. I nodi sensori della famiglia Telos considerati ormai lo standard sia dentro che fuori l'università hanno un costo che oscilla tra le 70 e le 100€ ed hanno il tremendo difetto di scalare molto lentamente rispetto alla tecnologia elettronica.

In questo lavoro pertanto si è deciso di costruire una rete di mote a partire da componenti COTS<sup>2</sup> utili per la rapida prototipazione, che rispondono al nome di STM32vdiscovery [35] e CC2520 [36]. La STM32vdiscovery è una board proposta dalla STMicroelectronics<sup>3</sup> a basso costo, pensata per introdurre l'utilizzatore nel mondo STM32. Il costo è irrisorio, non supera la decina di euro, ed ha un hardware del tutto comparabile con quello del Telosb (si veda la tabella 4.1)

	Stm32vdiscovery	Telosb
microcontrollore	ARM Cortex-M3 32 bit	TI MSP430 16 bit RISC
frequenza MCU	24MHz	1Mhz
ram	8KB ram	10KB ram
flash	128KB	48KB
adc	1 a 12bit 16 canali	1 a 12bit 8 canali
dac	1 a 12bit 2 porte	1 a 12bit 2 porte
alimentazione	1.8-3.36V, via USB	1.8-3.6V via USB
interfacce	SPI,IC2,UART,JTAG,DMA	SPI,I2C,UART,JTAG,DMA
timer	RTC 32 bit, 9 timer 16bit	2 timer 16bit
watchdog	2 a 12bit	15 a 16bit
consumo	14μA sleep, 3.5mA idle	5.1μA sleep, 1.8mA idle

Tabella 4.1: Comparazione STM32vdiscovery-Telosb

Il transceiver CC2520 della Texas Instruments è sostanzialmente un'evoluzione del CC24020 integrato nei Telosb. E' compatibile con la standard ZigBee/IEEE 802.15.4, lavora nella banda 2.4GHz è ultra low-power ed offre diverse funzioni hardware orientate al timing. La figura 4.2.2 dall'idea di come è fatto fisicamente

<sup>2</sup>Commercial Off-The-Shelf, si riferisce a componenti hardware e software disponibili sul mercato per l'acquisto da parte di aziende di sviluppo interessate a utilizzarli nei loro progetti.

<sup>3</sup><http://www.st.com/web/en/home.html>

il mote proposto. Ovviamente non essendo interessati a monitorare alcun reale fenomeno fisico ma semplicemente al collaudo di FLOPSYNC, l'unico sensore disposto a bordo mote è quello di temperatura.

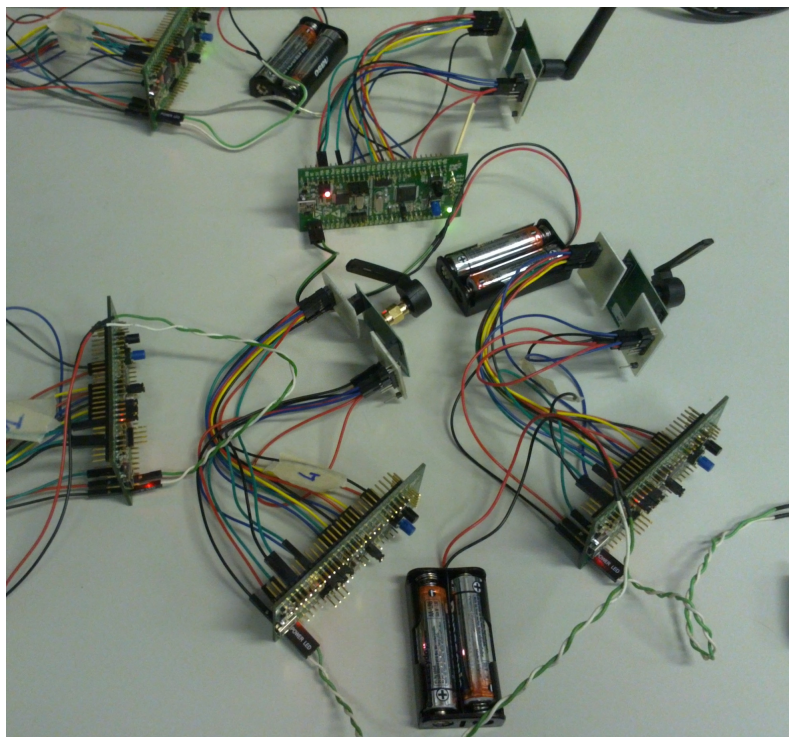


Figura 4.2.2: Alcuni mote usati per gli esperimenti.

Come sistema operativo è stato utilizzato Miosix [37] sviluppato sin dal 2008 all'interno del Politecnico di Milano. Nasce dall'idea di Federico Terraneo, praticamente unico sviluppatore del kernel, pensato per il mondo embedded ed è completamente open source rilasciato sotto licenza GPL. La filosofia che guida l'intero sviluppo del kernel Miosix e che lo distingue da quelli esistenti in questo campo poggia le sue fondamenta sulle seguenti:

- fornire un ambiente di sviluppo per quanto possibile “compliant standard”, coerente con le limitazioni hardware del microcontrollore al fine di ridurre il divario tra lo sviluppo di applicazioni desktop ed embedded;
- “Non si paga per ciò che non si usa”. Questo viene fatto quando possibile automaticamente oppure tramite le opzioni di configurazione.

Miosix è pensato per microcontrollori a 32 bit (famiglia LPC200, ARM7, ARM Cortex-M3, ecc...) è interamente scritto in C++ ed integra la possibilità di scrivere applicazioni in C++ utilizzando la libreria standard del C e la STL del C++, opportunamente riprogettate per sistemi embedded. Supporta il multithreading e

il multitasking e non necessita di MMU per farlo. La sua architettura modulare offre la possibilità di abilitare feature piuttosto avanzate ed allo stesso tempo di essere utilizzato con hardware con pochi KB di flash. E' adatto per applicazioni low-power e supporta lo sviluppatore con funzionalità di gestione della memoria ed informazioni dettagliate per il debug delle applicazioni.

### 4.3 Il livello MAC dello stack radio

Sfortunatamente Miosix prima di questo lavoro, non possedeva il supporto per il transceiver CC2520, pertanto il primo passo fatto verso l'implementazione di FLOPSYNC è stato quello di scrivere il driver per inviare e ricevere pacchetti. In questa sezione non intendiamo annoiare il lettore con configurazioni di registri e comunicazioni SPI<sup>4</sup>, che se ne ha voglia può benissimo trovare sia nella documentazione che nel codice sorgente, piuttosto l'idea è quella di mostrare in che modo FLOPSYNC sfrutta i servizi offerti da livello MAC dello stack radio mostrato in figura 3.4.1.

A livello MAC lo standard IEEE 802.15.4 [34] definisce il formato del frame nel seguente modo: 4 byte di preambolo, 1 byte di Start Frame Delimiter (SFD), 1 byte di Frame Length che indica il numero di byte dell'MPDU, l'MPDU che per il bsp sarà composta di solo 2 byte ossia un contatore per l'hop ed il suo negato per controllarne il checksum in ricezione. Il CC2520 incorpora parte dell'elaborazione del frame come ad esempio l'aggiunta in automatico del Synchronisation Header e del Frame Check Sequence (se lo si desidera).

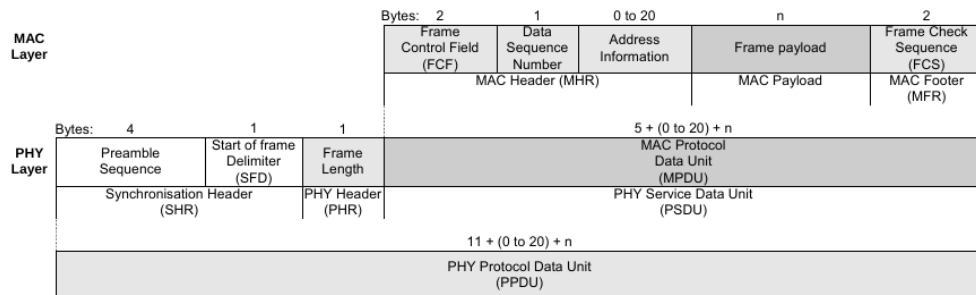


Figura 4.3.1: Vista schematizzata del formato del frame IEEE 802.15.4 [1]

La cosa davvero interessante ai fini della nostra trattazione è la possibilità da parte del CC2520 di ricevere degli interrupt per gestire accuratamente la temporizzazione di un frame. Ad esempio è possibile alzare un interrupt su un opportuno

<sup>4</sup>Serial Peripheral Interface è un sistema di comunicazione tra un microcontrollore e altri circuiti integrati o tra più microcontrollori ideato dalla Motorola.

GPIO<sup>5</sup> del CC2520 quando l'invio dello Start Frame Delimiter (SFD) è stato completato e la medesima cosa può essere fatta quando un SFD viene ricevuto. Inoltre anche quando l'invio o la ricezione del frame è stata completata può essere sollevato un interrupt che in questo caso è denominato Frame Done (FRM\_Done). L'utilizzo di quest'ultimo ad esempio potrebbe essere utile per leggere il frame ricevuto dal buffer di ricezione.

L'invio di particolari comandi (detti command strobe) oltre che tramite SPI possono essere inviati al CC2520 anche direttamente configurando un GPIO in ingresso associato a quel particolare comando. Questo meccanismo risulterà utile per fare l'invio hardware di un frame di cui si è avuto cura di memorizzare precedentemente nel buffer del transceiver.

Il protocollo FLOPSYNC, per questioni di ottimizzazione, interagisce direttamente con il livello MAC dello stack radio utilizzando funzionalità quali la ricezione di interrupt (come SFD, FRM\_DONE ecc...) e l'invio di comandi (ad esempio STXON per l'invio del pacchetto) altrimenti inutilizzabili. La possibilità di operare a questo livello permette anche di inviare pacchetti senza aggiunta di byte di header derivanti dai livelli superiori dello stack radio.

## 4.4 Un'efficiente implementazione di timer assoluto

Il conteggio del tempo e la sua precisione sono essenziali per realizzare un buon protocollo di sincronizzazione. Ogni microcontrollore integra al suo interno da due timer fino a qualche decina, allo scopo di misurare il tempo e generare interrupt in alcuni punti specifici del tempo. Tipicamente nelle WSN due sono i modelli di timer utilizzati: *assoluto* e *relativo*. La prima tipologia tiene il tempo da quando il nodo viene alimentato, la seconda tiene il tempo solo per brevi periodi. Possiamo paragonare il modello del timer assoluto ad un orologio e quello relativo ad un cronometro. Un timer assoluto ha di contro quello di essere molto più difficile da implementare rispetto ad un timer relativo ma ha il grosso vantaggio di essere più accurato se usato in uno schema di sincronizzazione. Di seguito viene mostrato come implementare un modello di timer assoluto in maniera efficiente specificando cosa s'intende per efficienza in questo caso.

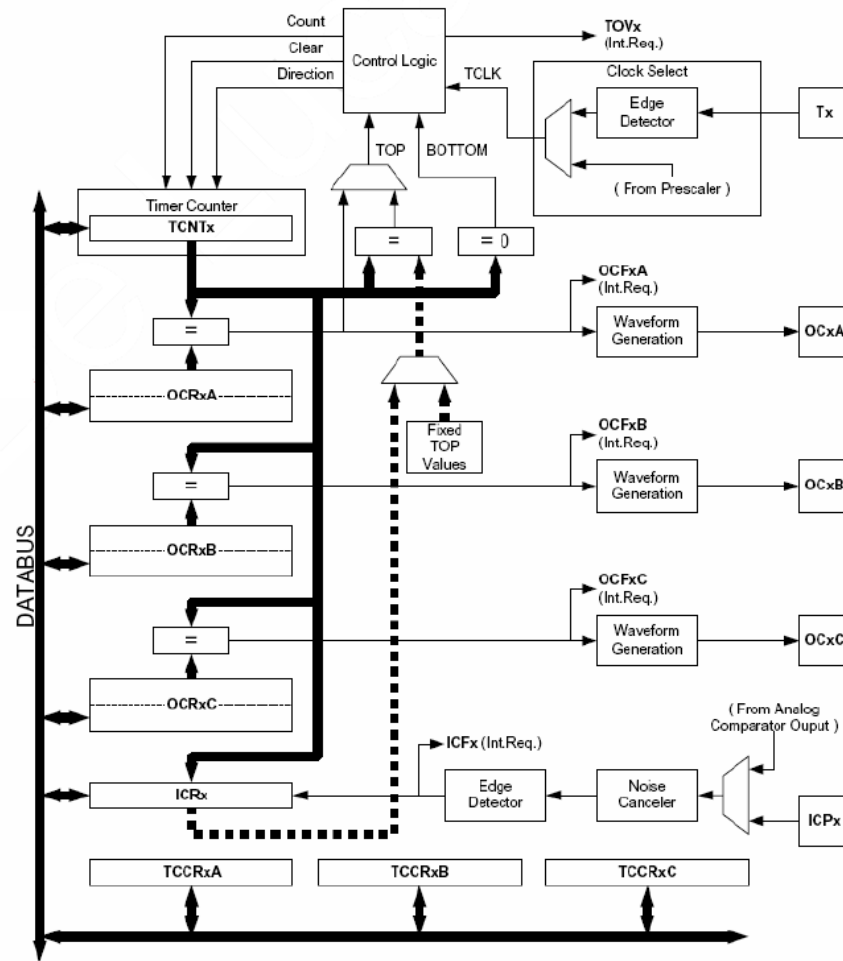


Figura 4.4.1: Schema a blocchi di un comune timer da microcontrollore.

#### 4.4.1 Timer: definizione ed uso nei microcontrollori

Il timer è un componente hardware, integrato nel microcontrollore, la cui funzione è quella di contare il numero di impulsi al suo ingresso. In un microcontrollore sono generalmente presenti più di un timer e possono essere utilizzati per gli usi più disparati. Un esempio di timer è schematizzato in figura 4.4.1. Ogni timer è costituito da i seguenti blocchi:

- *control logic*, è il blocco che permette di impostare le funzioni del timer come ad esempio la modalità di conteggio (in avanti, all'indietro, auto reload ecc..), la generazione di eventi o interrupt (overflow, underflow ecc...) ed avviare o fermare il conteggio;

<sup>5</sup>General Purpose Input/Output è un'interfaccia programmabile per inviare o ricevere segnali dal microcontrollore.



- *clock select*, è il blocco dedicato alla gestione del segnale di clock. Il clock può essere esterno o interno e la sua frequenza può essere scalata attraverso un prescaler;
- *timer counter*, costituito dal registro di conteggio;
- *capture/compare* (non sempre presente), contiene uno o più registri di *input capture* e/o di *output compare*. La funzione di input capture è utile per memorizzare il contenuto del contatore al verificarsi di un interrupt o di un evento, quella di output compare è la duale e permette di generare un interrupt o un evento attraverso la comparazione tra il registro di output capture ed il timer counter. Sfruttando opportunamente queste funzioni si possono generare anche forme d'onda particolari.

La dimensioni dei registri variano da timer a timer, generalmente sono a 16 o 32 bit.

#### 4.4.2 Il Real Time Clock

Quasi tutti i microcontrollori dispongono di un timer particolare chiamato *Real Time Clock (RTC)*. Come intuibile dal nome, questo dispositivo è indicato per funzioni di orologio e si differenzia dai precedenti perché è in grado di contare anche quando il microcontrollore va in modalità low-power. Per poter mantenere il conteggio del tempo, gli RTC si avvalgono di un oscillatore al quarzo (di solito con una frequenza di 32.768KHz) dai bassi consumi a loro dedicato. Solitamente ha una struttura più semplice rispetto a quella illustrata in figura 4.4.1 dove il blocco di capture/compare è assente o estremamente semplificato.

L'RTC è pensato per realizzare il modello di timer assoluto con un basso consumo di potenza. Un timer assoluto realizzato con un RTC a 32.768KHz ha una risoluzione di poco superiore ai  $30\mu s$ . Questo dato evidenzia subito i limiti di questa tipologia di timer: la *risoluzione*. Uno schema di sincronizzazione che utilizza un timer RTC nel modo appena descritto non può raggiungere un'accuratezza elevata. Pensare di agire sulla frequenza dell'RTC non è una buona scelta in quanto non migliora il *trade-off risoluzione-potenza*.

#### 4.4.3 Il Virtual High resolution Time

Una soluzione piuttosto interessante al problema risoluzione-potenza del timer assoluto, è stata proposta da *T. Schmid et al.* nel 2010 [23] e risponde al nome di

*Virtual High-resolution Time (VHT)*. L'idea che c'è alla base è quella di combinare assieme, in software, un timer *lento* ed uno *veloce* in modo da poter sfruttare i pregi dell'uno e dell'altro. Il funzionamento è piuttosto semplice: in modalità low-power, dove non è necessario avere una grande risoluzione, si utilizza il normale RTC, mentre in modalità power si utilizza un secondo timer ed un oscillatore al quarzo con una frequenza molto più elevata (in generale ogni sistema embedded è equipaggiato con almeno due quarzi come quelli descritti). Lo scotto da pagare per un clock combinato di questo tipo è semplicemente dovuto ad una implementazione più complessa del sistema orologio inteso come combinazione di quello lento e di quello veloce, in quanto saranno necessarie operazioni di sincronizzazione tra i due.

I dettagli che rendono realizzabile l'idea appena descritta possono essere trovati nell'articolo dell'autore [23] o anche visti all'interno di FLOPSYNC, dato che il timer è stato realizzato con il VHT. La cosa davvero interessante, che neppure l'autore menziona nel suo articolo è che, utilizzando un ulteriore quarzo per scandire il tempo oltre quello dell'RTC, le componenti di offset, skew e drift di quest'ultimo si sommano a quelle del quarzo dell'RTC in funzione del tempo di utilizzo del timer veloce. Per uno schema di sincronizzazione pensato per compensare il solo errore del quarzo introdotto dall'RTC, significa commettere un errore di stima. Tale errore è dovuto alla mancata considerazione di offset, skew e drift del secondo quarzo. Dato che l'errore commesso dal secondo quarzo varia sensibilmente in funzione del tempo di utilizzo del timer veloce (d'altronde solo quando lo si utilizza si introduce questo tipo di errore) risulta difficile da compensare. Un esempio utile a chiarire le idee è il seguente:

supponiamo che il periodo di sincronizzazione sia un minuto e che subito dopo l'ultima sincronizzazione il nodo vada in modalità low-power, utilizzando quindi il clock lento per contare. Ipotizziamo che a metà del periodo di sincronizzazione il nodo si risvegli dalla modalità low-power per compiere delle operazioni e che resti in modalità power fino alla prossima sincronizzazione del protocollo. Al momento del risveglio il VHT avvia il conteggio del timer veloce, sincronizza timer lento e timer veloce e da questo momento, fino al prossimo evento che lo porterà il nodo in modalità low-power, il VHT discriminerà il tempo con il timer veloce. All'arrivo del pacchetto di sincronizzazione, l'errore commesso dal clock locale sarà per metà dovuto al timer lento e per metà a quello veloce. Questo esempio evidenzia che l'errore di sincronizzazione ha una componente fortemente dipendente dal tempo in cui il nodo resta in modalità power.

Risultati sperimentali hanno mostrato che utilizzando un VHT implementato come suggerito nel [23], a fronte di un conteggio per soli cento millisecondi operato con il timer veloce prima dell'arrivo del pacchetto di sincronizzazione, ag-

giunge un errore di  $\pm^{200}/f_{clock-veloce}[s]$ . L'esperimento è stato condotto utilizzando un oscillatore a 24MHz con variazione di 75ppm per il timer veloce.

L'utilizzo di un VHT siffatto è di scarsa utilità in quanto il sistema escogitato per migliorare l'accuratezza dello schema di sincronizzazione finisce addirittura per peggiorarla.

La soluzione proposta con questo lavoro (implementata in FLOPSYNC), al fine di poter utilizzare il VHT, è quella che illustriamo di seguito. Il contributo di errore introdotto dal clock veloce non si può evitare che compaia, può però essere limitato ed addirittura essere considerato inesistente se minore di  $\pm^1/f_{clock-veloce}[s]$  in quanto al di sotto della risoluzione del VHT. L'idea per cui è quella di re-sincronizzare periodicamente RTC e timer veloce in modo tale da azzerare il contributo d'errore del secondo ad ogni sincronizzazione. Quanto frequenti debbano essere le re-sincronizzazioni può essere deciso in base all'accuratezza desiderata ed alla variazione di frequenza del clock veloce espressa in parti per milione. Giusto per dare un'idea, per l'architettura proposta, re-sincronizzando il timer veloce ogni 20 tick dell'RTC si ha un errore inferiore ad un tick del timer veloce. Chiaramente le re-sincronizzazioni del VHT sono necessarie solo quando si usa il timer veloce e non in modalità low-power.

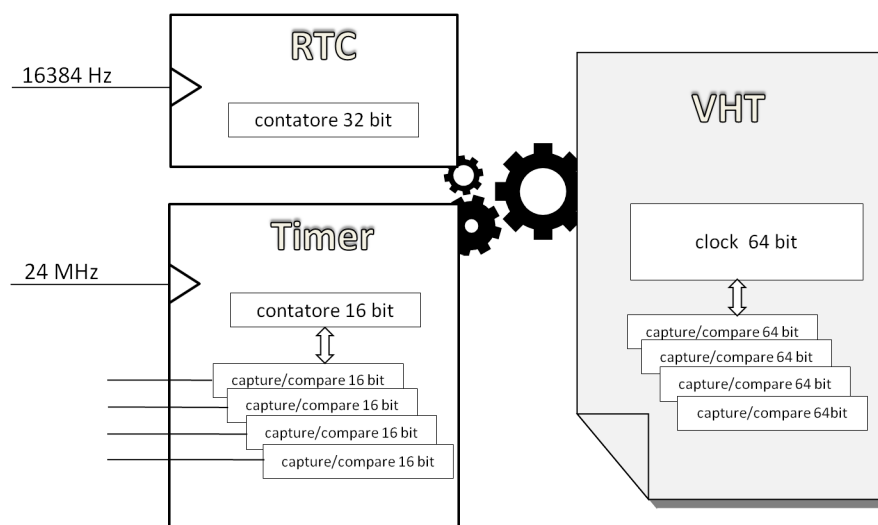


Figura 4.4.2: Modello del timer.

Il modello di timer proposto pertanto è quello raffigurato in figura 4.4.2. Riepilogando, il VHT è un timer virtuale (implementato completamente in software) che si basa sull'RTC e su un secondo timer hardware come spiegato in precedenza. Il VHT offre la possibilità di generare e catturare interrupt o eventi attraverso comparazioni tra i registri virtuali di capture/compare ed il proprio clock (o contatore virtuale che dir si voglia). Il VHT è un timer assoluto in quanto il contatore

virtuale tiene il tempo a partire da quando il nodo è stato alimentato e le operazioni di capture/compare vengono fatte sul tempo assoluto. Il vantaggio principale, rispetto ad un timer relativo, è il disaccoppiamento tra il tempo in cui si setta il timer e la gestione degli eventi o interrupt. Per chiarire le idee, si supponga ad esempio di voler generare un interrupt esattamente ogni 60 secondi. Per il timer modellato occorre semplicemente scrivere nel registro virtuale di compare il valore di  $60 * f_{VHT}$  per generare il primo interrupt, per quello successivo  $120 * f_{VHT}$  e così via. Ai fini della precisione con cui si genera l'interrupt non ha importanza il “quando” viene scritto il registro virtuale di compare, l'importante è che non venga fatto dopo che il clock del VHT superi il tempo scritto nel registro. Si ipotizzi di voler fare la medesima operazione con un timer relativo, non essendoci più il concetto di tempo assoluto, per poter generare l'interrupt esattamente ogni 60 secondi è importante che l'avvio del timer venga fatto in maniera estremamente precisa in quanto essendo relativo ogni ritardo o anticipo nell'avvio si riflette sull'interrupt.

Il concetto di timer assoluto ha come contro la necessità di dover utilizzare il VHT che richiede oltre un overhead di sincronizzazione tra timer lento e veloce anche un contatore virtuale e registri di capture/compare virtuali a 64 bit per evitare che vadano in overflow dopo breve tempo.

## 4.5 Temporizzazione del bsp

L'accuratezza dell'intero protocollo di sincronizzazione verte sul meccanismo temporizzato di invio/ricezione del bsp. Questa sezione mostra come raggiungere temporizzazioni accurate del bsp fondendo assieme le funzioni offerte dal timer con quelle del transceiver.

Si è detto nella sezione 4.3 che il transceiver dispone di *command strobe*, ossia di comandi attuabili tramite un segnale su opportuno GPIO. Tra questi comandi vi è quello per iniziare la trasmissione di un frame contenuto nel buffer. Si è detto anche che il VHT è in grado di generare un interrupt quando il clock eguaglia il valore contenuto in uno dei registri capture/compare virtuali. Unendo queste due funzionalità si può generare l'invio in hardware del bsp in maniera estremamente accurata.

Il timestamp del bsp, ossia l'istante di tempo in cui il bsp viene ricevuto, può essere ottenuto sfruttando il meccanismo di capture del VHT e unito alla capacità del transceiver di generare interrupt quando la ricezione dell'SFD è stata completata. Sapendo che il ritardo di trasmissione e la sua incertezza sono quelli calcolati con la 5.1.1 e 5.1.2 si ha la conoscenza del tempo d'invio del bsp.

L'incertezza del ritardo di trasmissione è il vero percorso critico che limita l'accuratezza della sincronizzazione.

## 4.6 Cenni implementativi del controllore

Per dimostrare la semplicità del controllore FLOPSYNC e della finestra dinamica si riporta di seguito la completa implementazione C++.

```

class FLOPSYNC : public Synchronizer
{
public:
    /**
     * Constructor
     */
    FLOPSYNC();

    /**
     * Compute clock correction and receiver window given synchronization error
     * \param e synchronization error
     * \return a pair with the clock correction, and the receiver window
     */
    std::pair<int,int> computeCorrection(int e);

private:
    int uo, uoo;           //Past control action
    short eo, eoo;        //Past error samples
    int sum, squareSum;   //Variance computation data
    unsigned char count; //Variance samples counter
    unsigned char dw;     //Computed window size
    char init;           //Controller preinitialization

    static const int numSamples=8; //Number of samples for variance computation
    static const int fp=64;
    //Fixed point, log2(fp) bits are the decimal part
    //The maximum value that can enter the window computation algorithm without
    //without causing overflows is around 700, resulting in a scaleFactor of
    //5 when the vht resolution is 1us, and w is 3ms. That however would cause
    //overflow when writing the result to dw, which is just an unsigned char
    //(to save RAM). This requires a higher scale factor, of about w/255, or 12.
    //However, this requires more iterations to approximate the square root,
    //so we're using a scale factor of 30.
    static const int scaleFactor=480;
};

FLOPSYNC::FLOPSYNC()
{
    uo=uoo=eo=eoo=sum=squareSum=count=0;
    dw=w/scaleFactor;
    init=0;
}

pair<int,int> FLOPSYNC::computeCorrection(int e)
{
    //Controller preinit, for fast boot convergence

```

```

switch(init)
{
    case 0:
        init=1;
        eo=e;
        uo=2*512*e;
        uoo=512*e;
        return make_pair(2*e,w); //One step of a deadbeat controller
    case 1:
        init=2;
        eo=0;
        uo/=2;
}

//Flopsync controller, with alpha=3/8
//u(k)=2u(k-1)-u(k-2)-1.875e(k)+2.578125e(k-1)-0.947265625e(k-2) with values kept multiplied by 512
int u=2*uo-uoo-960*e+1320*eo-485*eo;
uoo=uo;
uo=u;
eo=eo;
eo=e;

int sign=u>=0 ? 1 : -1;
int uquant=(u+256*sign)/512;

//Scale numbers if VHT is enabled to prevent overflows
int wMax=w/scaleFactor;
int wMin=minw/scaleFactor;
e/=scaleFactor;

//Update receiver window size
sum+=e*fp;
squareSum+=e*e*fp;
if(++count>=numSamples)
{
    //Variance computed as E[X^2]-E[X]^2
    int average=sum/numSamples;
    int var=squareSum/numSamples-average*average/fp;
    //Using the Babylonian method to approximate square root
    int stddev=var/7;
    for(int j=0;j<3;j++) if(stddev>0) stddev=(stddev+var*fp/stddev)/2;
    //Set the window size to three sigma
    int winSize=stddev*3/fp;
    //Clamp between min and max window
    dw=max<int>(min<int>(winSize,wMax),wMin);
    sum=squareSum=count=0;
}

return make_pair(uquant,scaleFactor*dw);
}

```

Come si può notare dal codice, l'intero algoritmo è implementato in aritmetica intera. Anche la radice quadrata per il calcolo della deviazione standard è svolto con aritmetica intera e si basa sul metodo Babilonese [38]. Per l'architettura proposta l'algoritmo richiede solamente 24 byte di RAM per l'istanza della classe (più pochi

byte per lo stack) e la classe completa (inclusi i metodi non mostrati qui per brevità) richiede solamente 604 byte di memoria. Per quanto riguarda la velocità di esecuzione della funzione *computeCorrection()* normalmente impiega  $4.4\mu s$ , quando invece viene ricalcolata la finestra allora impiega  $7.2\mu s$ .





# Capitolo 5

## Risultati sperimentali

Questo capitolo è dedicato alla valutazione delle performance del protocollo di sincronizzazione FLOPSYNC. Di seguito verranno riportate alcune prove sperimentali significative, volte a misurare le prestazioni di FLOPSYNC ed altre comparative. L'architettura di riferimento è quella descritta nel capitolo 4.2. In tutti gli esperimenti riportati il periodo di sincronizzazione  $T$  è pari a  $60s$  ed  $\alpha$  è  $3/8$  se non diversamente specificato. Per misurare l'errore di sincronizzazione all'arrivo del bsp, il nodo sincronizzato piuttosto che ricevere il pacchetto di sincronizzazione e subito dopo rinviarlo, invia periodicamente un pacchetto addizionale (contiene anche informazioni aggiuntive come il valore della finestra  $w$  piuttosto che la variabile di controllo  $u$ ) al nodo reference. Questo pacchetto è schedulato usando un approccio TDMA (più precisamente si utilizza un TDMA sincrono con uno slot pari a  $0.5s$ ; ad ogni nodo della rete è associato un preciso slot configurato in base al numero di nodi che compongono la rete) per prevenire collisioni e il tempo quando il pacchetto deve essere inviato è calcolato usando la formula della *traslazione all'indietro* del clock virtuale (3.3.5). Il nodo reference, non appena riceve il pacchetto, calcola la differenza tra il tempo di arrivo *expected* e quello *actual* e lo salva in un file di log. I dati riportati di seguito sono stati collezionati in questo modo.

### 5.1 Quantificazione del jitter indotto dall'hardware

Una prima campagna di esperimenti, come suggerito in [23], è volta a quantificare l'accuratezza dell'hardware in termini di sincronizzazione. Data una piattaforma hardware, la qualità di sincronizzazione raggiungibile non può essere migliore del jitter della linea di interrupt del pacchetto ricevuto. Due prove sperimenta-

li vengono riportate di seguito per quantificare l'accuratezza d'invio/ricezione del transceiver.

Premesso che in letteratura non vi è nulla a riguardo della precisione del meccanismo di interrupt del CC2520, si è deciso di misurarlo. La figura 5.1.1a illustra una sequenza di misurazioni eseguite su tre nodi con CC2520 per caratterizzare l'accuratezza degli interrupt. In questo esperimento un nodo trasmette e l'altro riceve un frame (non ha importanza il contenuto del payload). L'obiettivo è quello di misurare con un oscilloscopio il ritardo di tempo tra il fronte di salita dell'SFD trasmettitore e il tempo di salita SFD ricevitore. Il medesimo esperimento è stato ripetuto con due nodi differenti in ricezione. Quello che si può osservare è che le distribuzioni sono quasi identiche con in media un ritardo di  $3.3\mu s$  circa ed un'incertezza pari a  $0.04\mu s$  circa.

La figura 5.1.1b è il risultato di un secondo esperimento avente questa volta come obiettivo quello di caratterizzare il ritardo tra il fronte di salita con cui si comanda al transceiver l'invio di un frame ed il fronte di salita in cui lo stesso transceiver genera l'interrupt SFD. Il risultato di questo esperimento è un tempo medio di  $352.37\mu s$  con un'incertezza pari a  $0.004\mu s$ . Il tempo medio calcolato può essere visto come la somma per inviare 5 byte di SHR ( $160\mu s$  con data rate di 250 Kb/s) più il tempo impiegato dal transceiver per calibrare la frequenza di trasmissione, detto *TX turnaround time*. Il *TX turnaround time* riportato sul datasheet è pari a  $192\mu s$ , in realtà da quanto emerge da questo esperimento possiamo affermare che mediamente ne impiega circa  $192,37\mu s$ .

A questo punto sorge spontanea la domanda: "a quanto ammonta il ritardo complessivo tra i fronti di salita del segnale di trigger di trasmissione e quello di interrupt SFD in ricezione?". Indichiamo con  $d_{sTX-sRX}$  il ritardo medio tra i fronti di salita degli interrupt di SFD in trasmissione e in ricezione, e con  $d_{t-sTX}$  il ritardo medio tra i corrispettivi fronti di salita del trigger e dell'SFD entrambi in trasmissione. Indichiamo con  $u$  le corrispettive incertezze dei ritardi appena definiti. Per cui troviamo che il ritardo complessivo tra trigger ed SFD in ricezione è pari a:

$$d_{t-sRX} = d_{t-sTX} + d_{sTX-sRX}, \quad (5.1.1)$$

mentre la sua incertezza (supponendo scorrelate le due misure) è data da:

$$u(d_{t-sRX}) = \sqrt{\left[\frac{\partial d_{t-sRX}}{\partial d_{t-sTX}}\right]^2 u^2(d_{t-sTX}) + \left[\frac{\partial d_{t-sRX}}{\partial d_{sTX-sRX}}\right]^2 u^2(d_{sTX-sRX})}. \quad (5.1.2)$$

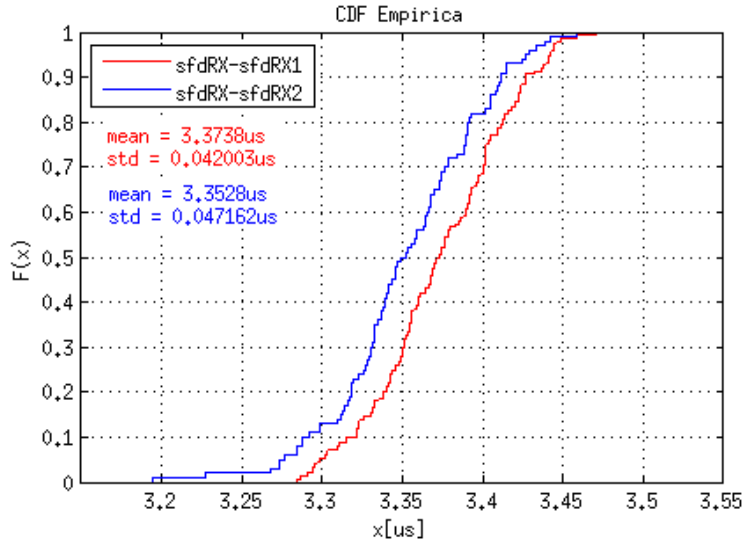
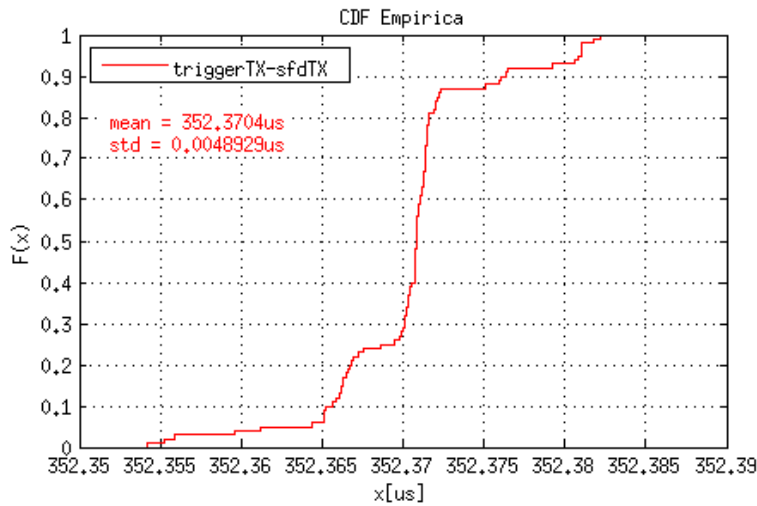
(a) Tempi tra SFD [ $\mu$ s].(b) Tempo tra trigger TX e SFD TX [ $\mu$ s].

Figura 5.1.1: Distribuzioni empiriche cumulative tra i tempi di interrupt del transceiver CC2520.

Sostituendo i valori numerici relativi alle sole curve in rosso di figura 5.1.1, alle formule 5.1.1 e 5.1.2 si ottiene il ritardo complessivo e la sua incertezza pari a  $d_{t-sRX} = 355.7442\mu s$  e  $u(d_{t-sRX}) = 0.042\mu s$ .

Il risultato appena ottenuto ci porta a concludere che ad un livello di confidenza del 99.7% l'incertezza del ritardo complessivo è di circa  $126ns$ . Questo risultato si traduce in una limitazione delle cifre significative dello schema di sincronizzazione. Quindi pur disponendo di un clock con risoluzione maggiore di  $126ns$ , le cifre significative dell'errore di sincronizzazione non possono varcare questa soglia.

## 5.2 Valutazione delle performance di FLOPSYNC

In questa sezione saranno valutate le performance di sincronizzazione in senso stretto. Il primo esperimento mostra il comportamento del protocollo al boot del nodo da sincronizzare, mentre il secondo mostra l'errore di sincronizzazione a regime.

### 5.2.1 Boot del nodo

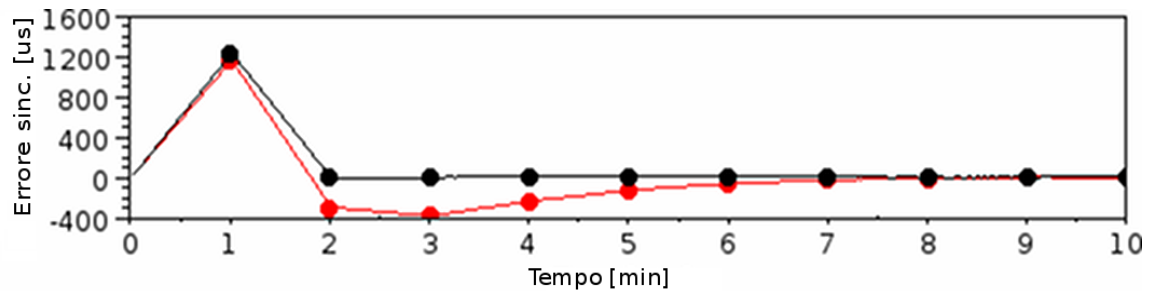


Figura 5.2.1: Errore di sincronizzazione durante la fase di boot di FLOPSYNC, con il controllore deadbet (in nero) e senza in (rosso).

Questo esperimento ha lo scopo di valutare le prestazioni di risposta al gradino del controllore FLOPSYNC e del clock virtuale. Il comportamento che andremo a testare è rappresentativo sia quando il nodo viene avviato per la prima volta e non ha alcuna informazione sullo skew, sia quando si re-sincronizza a seguito di una desincronizzazione causata ad esempio da interferenze radio. In questo esperimento un nodo avente un clock skew di  $20ppm$  viene avviato ed il nodo reference memorizza l'errore di sincronizzazione nei primi minuti di funzionamento. Il risultato di figura 5.2.1 mette a confronto l'errore di sincronizzazione di FLOPSYNC con controllore deadbeat di preinizializzazione (linea nera) e senza (linea rossa).

Come si può vedere, e compatibilmente con le aspettative di sintesi del controllo, il controllore può orientare l'errore di sincronizzazione verso lo zero riducendolo in valore assoluto a meno dell'1% di quello originario, in soli nove periodi di sincronizzazione senza preinizializzazione. Quando viene aggiunto il controllore deadbeat per la preinizializzazione, il tempo richiesto per la convergenza è ridotto a soli due minuti. Per quanto riguarda il clock virtuale la sua monotonia è garantita dall'assenza di salti, ottenendo pertanto un errore di sincronizzazione che si raccorda "dolcemente" tra  $e_i(t(k)^-)$  ed  $e_i(t(k)^+)$  e si assesta ad un valore prossimo a zero quando il controllore raggiunge il suo stato di regime. Si noti che la variazione di segno dell'errore di sincronizzazione che avviene quando il controllore deadbet è di-

sabilitato non causa problemi di monotonicità, semplicemente significa che il clock passa da un frequenza maggiore del reference ad una minore, prima di convergere.

### 5.2.2 Distribuzione statistica dell'errore di sincronizzazione

Questo esperimento ha come obiettivo ultimo quello di verificare la distribuzione dell'errore di sincronizzazione del protocollo FLOPSYNC eseguito in una rete di sensori wireless composta da più hop. I nodi sono stati distribuiti all'interno di un edificio ed esposti alle interferenze della rete wi-fi locale, come accade in una situazione reale, per un tempo complessivo di 6 giorni consecutivi.

	Errore di tutto il set di dati		$e(t(k+1)^-)$		Finestra ( $w$ )	
	Media	Dev. Std.	Media	Dev. Std	Media	Dev. Std.
Primo hop	50ns	866ns	298ps	995ns	20.17μs	3.19μs
Secondo hop	26ns	858ns	-817ps	988ns	20.19μs	3.42μs
Terzo hop	-25ns	858ns	333ps	999ns	20.46μs	5.28μs
Quarto hop	-12ns	877ns	-57ps	1025ns	20.43μs	5.09μs
Quinto hop	29ns	873ns	403ps	1020ns	20.66μs	6.27μs
Sesto hop	-62ns	880ns	452ps	1027ns	20.88μs	7.74μs
Settimo hop	-113ns	891ns	-465ps	1042ns	21.08μs	10.47μs
Ottavo hop	-104ns	888ns	-23ps	1015ns	26.33μs	157.66μs
Media	-26ns	874ns	15ps	1014ns	21.28μs	24.89μs

Tabella 5.1: Media e deviazione standard dell'errore e della finestra di sincronizzazione per un esperimento multi-hop con periodo di 60s e durata di 6 giorni.

La tabella 5.1 riporta media e varianza per: l'errore di sincronizzazione complessivo (ossia calcolato come spiegato ad inizio capitolo), l'errore calcolato un istante prima della sincronizzazione ed il comportamento della finestra di sincronizzazione, tutti in funzione degli hop. La prima cosa da notare è il valore medio estremamente basso dell'errore un istante prima della sincronizzazione, pochi *picosecondi*, un ordine di grandezza in meno se confrontato con la risoluzione del timer VHT (un timer tick è circa 41ns). Questo è merito dell'integratore contenuto nel controllore FLOPSYNC che calcola la correzione del clock ed istantaneamente lo rimuove, secondo l'approccio errore a regime nullo. Questo è un vantaggio intrinseco per uno schema di sincronizzazione che si basa sulla teoria del controllo, al contrario di uno basato sulla regressione che è invece destinato a calcolare la correzione del clock sulla base di una finestra limitata di campioni. L'errore medio considerando l'intero dataset è invece più grande, anche se ancora al di sotto del microsecondo, la causa va attribuita al jitter della trasmissione radio.

Per quanto riguarda la deviazione standard, il valore assunto ad  $e_i(t(k+1)^-)$  è maggiore rispetto a quello dell'intero set di dati, in quanto questo è il momento più lontano dall'ultima sincronizzazione, ed è quindi quello più colpito dalle variazioni di breve periodo del quarzo, quali ad esempio la temperatura.

Se si osserva invece, la colonna relativa alla finestra di ricezione  $w$ , si nota subito come il meccanismo di adattamento porta ad avere in media valori di poco superiori al minimo imposto per la finestra di sincronizzazione (finestra minima  $20\mu s$ ). Analizzando con occhio critico la varianza di  $w$  e ricordando quanto detto nella sottosezione 3.4.2 se ne deduce che, le re-sincronizzazioni sono inesistenti o infrequenti e quindi il protocollo request/response per la compensazione dell'offset spiegato nella stessa sottosezione è una soluzione efficiente rispetto ad includere il timestamp in ogni pacchetto.

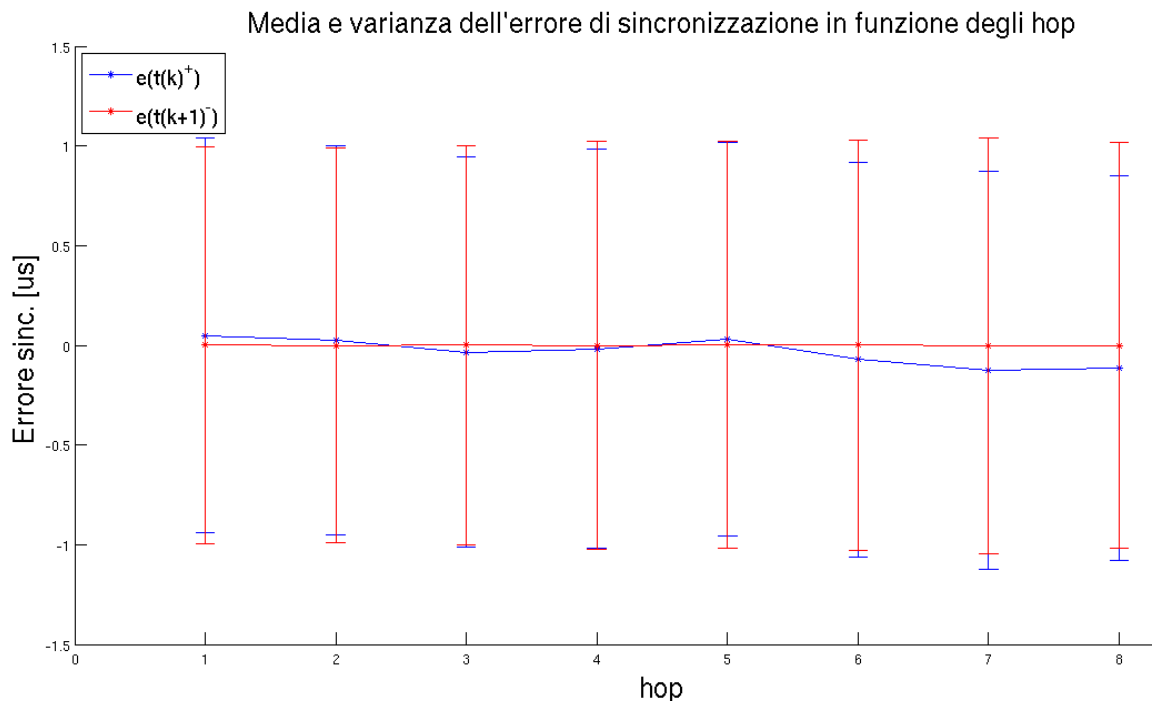


Figura 5.2.2: Media e varianza per l'errore un istante prima ed un istante dopo la sincronizzazione, in funzione degli hop.

La figura 5.2.2 mostra un interessante risultato ottenuto con questo esperimento. Per ogni hop sono riportati media e varianza dell'errore sia un istante prima della sincronizzazione che un istante dopo. Quello che salta subito all'occhio è l'indipendenza dell'errore di sincronizzazione dal numero di hop e lo si nota dall'uniformità delle barre orizzontali che esprimono la varianza dell'hop  $i$ -esimo. La motivazione di ciò è prettamente legata ad una implementazione corretta dei meccanismi di trasmissione/ricezione del bsp che, a differenza delle altre oggi esistenti, sfrutta a pieno le funzioni dell'hardware per migliorare la precisione complessiva.

La bontà di quanto affermato la si può apprezzare da un rapido confronto con quanto mostrato dalla figura 13 di [25]. Pertanto si dimostra che basare lo schema di sincronizzazione su meccanismi di temporizzazione imprecisi, come può esserlo l'invio di un pacchetto di sincronizzazione schedulato in software, si ripercuote amplificato sull'intero protocollo.

Interessante notare anche, come grazie al meccanismo di compensazione ed alla monotonia, la varianza dell'errore ad inizio sincronizzazione e quella a fine sincronizzazione siano circa uguali.

## 5.3 Comportamento sotto variazioni di temperatura

I risultati sperimentali riportati in questa sezione sono volti a mostrare la capacità di rigettare i disturbi di temperatura del controllore FLOPSYNC. Pertanto presenteremo un primo test comparativo con altri schemi di sincronizzazione ed un secondo focalizzato sulla robustezza del tool di configurazione FLOPSYNC.

### 5.3.1 Comparazione tra FLOPSYNC, FTSP e FBS

Come già detto nella sezione 3.2.2 la componente del disturbo più importante e difficile da rigettare è quella termica dovuta alla dipendenza parabolica del quarzo dalla temperatura. Ogni buon schema di sincronizzazione per essere denominato tale deve superare il cosiddetto test “transizione ombra-sole” [23] che causa drift di frequenza. Questa è una tipica situazione incontrata da un nodo usato in una applicazione WSN outdoor. In questo caso la temperatura del nodo cambia significativamente in un breve periodo causando continue variazioni nello skew che devono essere in qualche modo seguite dalla sincronizzazione fino a regime.

Per questo esperimento sono stati utilizzati tre nodi con tre schemi di sincronizzazione differenti: FLOPSYNC, FTSP ed FBS. I tre i nodi sono stati posti in un contenitore per componenti elettroniche ed spostati al sole. Il contenitore ha il duplice scopo: quello di fornire un impostazione realistica per un'applicazione esterna dato che il nodo deve resistere alle condizioni ambientali tipo pioggia e quello di mantenimento della temperatura in modo che sia il più uniforme possibile al fine di confrontare meglio la loro risposta. FTSP è stato configurato con una finestra di 8 campioni di dati, come nell'implementazione TinyOS, mentre FBS è stato configurato con  $K_i = K_p = 0,7847$ , come suggerito dagli autori.

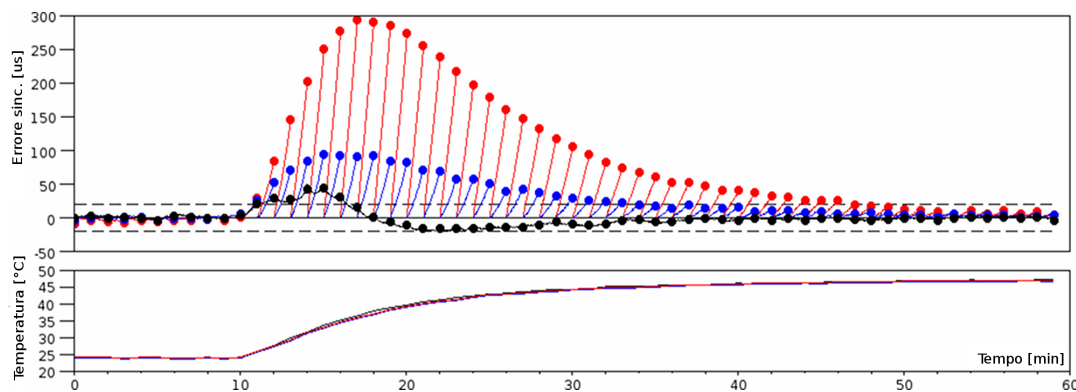


Figura 5.3.1: Comparazione tra FLOPSYNC (in nero), FTSP (rosso) e FBS (blu) in una transizione ombra-sole. I punti evidenziano l'errore ad  $e_i(t(k+1)^-)$ , mentre le linee mostrano l'errore di sincronizzazione all'interno di un periodo di sincronizzazione. La linea orizzontale tratteggiata delimita l'area dove l'errore è all'interno di  $20\mu s$ . I nodi sono esposti al sole al minuto 10.

Il risultato è mostrato in figura 5.3.1, dove il grafico superiore mostra la comparazione tra FLOPSYNC, FTSP ed FBS, mentre l'altro mostra la temperatura dei nodi. Se confrontato con FTSP, FLOPSYNC ottiene una riduzione significativa dell'errore di sincronizzazione, grazie al funzionamento in anello chiuso può compensare rapidamente lo skew. L'errore massimo osservato per FLOPSYNC è  $45\mu s$ , mentre con FTSP errori piuttosto alti come  $293\mu s$  si sono manifestati. Inoltre, FLOPSYNC richiede solo sette minuti dall'inizio della variazione di temperatura per limitare l'errore entro i  $20\mu s$ , mentre FTSP richiede 38 minuti.

FLOPSYNC e FBS sono progettati seguendo lo stesso approccio controllistico, e quindi entrambi possono limitare l'errore causato drift. Tuttavia, lo schema proposto, grazie alla sua struttura di controllo più avanzata, dopo aver raggiunto il massimo errore nei primi istanti della variazione di temperatura comincia a compensare, ed entro sette minuti può ridurre l'errore ad un valore accettabile di  $\pm 20\mu s$  e tenerlo in tale intervallo mentre la temperatura è ancora in aumento. Al contrario, la struttura di controllo PI (Proporzionale Integrale) di FBS presenta un maggior errore con un massimo di  $94\mu s$ , ma soprattutto, non compensa adeguatamente l'errore finché la temperatura non ha raggiunto uno stato stazionario. Il risultato è che impiega 28 minuti per ridurre l'errore nella gamma  $\pm 20\mu s$ . Per quanto riguarda le questioni di monotonicità del clock, si ricorda che, sia FTSP che FBS sono sistemi che fanno la sola compensazione dello skew e si basano sulla trasmissione di timestamp con correzione istantanea del clock. Con questo esperimento si chiude il cerchio aperto con l'esempio motivante della sezione 2.6 e dovrebbe rendere ancora più chiara la differenza tra uno schema con sola compensazione ed uno che combina sincronizzazione e compensazione di skew/drift



e garantisce la monotonicità del clock per progettazione. Sebbene FTSP ed FBS siano in grado di compensare uno skew costante quando la temperatura non cambia e quindi senza salti nel clock che causano problemi di monotonia, questo non è vero sotto variazioni di temperatura dove si riscontrano correzioni del clock locale nel passato.

### 5.3.2 Efficacia del tool di configurazione FLOPSYNC

Con questo esperimento si è voluto dimostrare l'efficacia e la robustezza del tool di configurazione FLOPSYNC anche sotto variazioni di temperatura e vincoli piuttosto ardui e surreali.

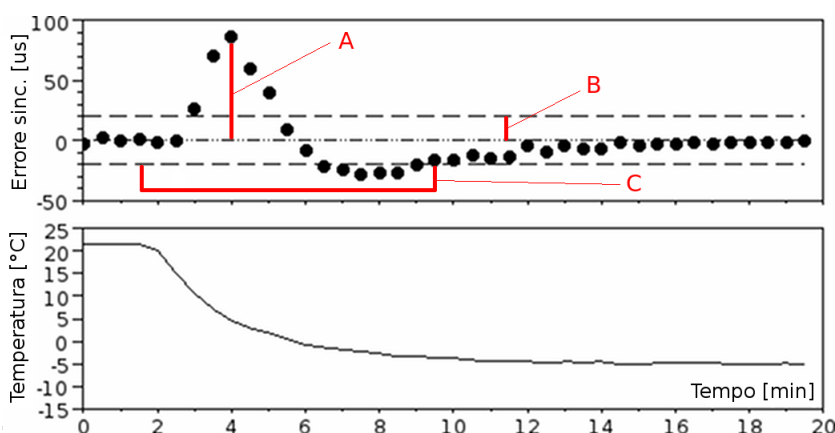


Figura 5.3.2: Errore di sincronizzazione durante una variazione di temperatura.

L'esperimento condotto si rifà al caso 1 mostrato nella sottosezione 3.2.5 in cui si ipotizza uno scenario ambientale con possibili variazioni di temperatura che variano tra un minimo di  $-20^{\circ}\text{C}$  ad un massimo di  $50^{\circ}\text{C}$  ed un coefficiente di massima variazione termica pari  $8^{\circ}\text{C}/\text{min}$ . I vincoli imposti sono invece:

- un errore massimo a fronte della variazione di temperatura inferiore a  $250\mu\text{s}$ ,
- un errore accettabile dopo la variazione di temperatura compreso tra  $\pm 20\mu\text{s}$ ,
- un tempo di recovery dell'errore inferiore a  $600\text{s}$ ,

rispettivamente indicati in figura 5.3.2 con le lettere A, B e C.

Inserendo questi parametri e quelli del quarzo decritti nello scenario del caso 1 nel tool di configurazione ed eseguendolo otteniamo l'insieme di coppie ammissibili rappresentate in un diagramma cartesiano (come quello di figura 4.1.2). Per l'esperimento la scelta della coppia  $T, \alpha$  è ricaduta su  $30\text{s}$  e  $3/8$  rispettivamente.

L'esperimento è stato condotto nel seguente modo: dopo la fase di boot si è lasciato che il controllore andasse a regime a temperatura ambiente, quindi dopo

alcuni periodi di sincronizzazione il nodo è stato posto in un ambiente con temperatura di circa  $-5^{\circ}C$  subendo un'escursione termica di poco inferiore a  $30^{\circ}C$  con una variazione di  $8^{\circ}C/min$ . Inoltre, come si può notare dalla figura 5.3.2, il nodo è stato esposto al cambiamento di temperatura un istante dopo l'evento di sincronizzazione, ossia la situazione peggiore che si potesse verificare in quanto tutta la dinamica del disturbo si concentra mentre il controllore è in anello aperto. L'errore massimo di sincronizzazione osservato è di  $86\mu s$  e come atteso inferiore a limite di  $250\mu s$ . Per quello che riguarda il tempo di recovery all'interno di  $20\mu s$  il nodo impiega 7 minuti, ancora una volta siamo ben al disotto del vincolo imposto.

# Capitolo 6

## Modello del consumo di potenza

Questo capitolo fornisce i dati del consumo di potenza al fine di provare l'efficienza energetica di FLOPSYNC. La misurazione del consumo di potenza intesa come l'operazione di misura fatta con il wattmetro è tutt'altro che semplice. In casi come questo l'approccio ingegneristico suggerisce di avvalersi dell'aiuto dei modelli.

Prima di presentare il modello di potenza è necessario che il lettore abbia un'idea degli stati operativi della CPU e della radio. A tale scopo si guardino le figure 6.0.1a e 6.0.1b. Le macchine a stati riportate per il microcontrollore e per la radio sono una versione semplificata di quelle realmente implementate, ma più che sufficienti per la comprensione di quanto segue. Ad ogni stato operativo si può associare un consumo di corrente generalmente calcolato dal costruttore e riportato nel datasheet. Il consumo di potenza si può calcolare indirettamente a partire dal consumo di corrente.

Con la premessa appena fatta possiamo quindi modellare il consumo di potenza di FLOPSYNC profilando le transizioni tra gli stati operativi di CPU e radio. Questo in pratica si traduce nella misurazione, del tempo speso in ogni stato operativo del controllore e del transceiver. Al fine di misurare il tempo in cui si permane in ciascuno stato, all'interno del codice che implementa FLOPSYNC è stato inserito un opportuno livello di debug denominato *probe debug*, con la funzione di associare la transizione di stato  $X$  al GPIO  $X$  reso osservabile dall'esterno sotto forma di spike<sup>1</sup>. Misurando con l'oscilloscopio il tempo dei fronti di salita tra uno stato e l'altro si può costruire il modello del tempo trascorso in ogni stato, sia per il nodo reference che per quello generico. Se al modello appena costruito si associano i consumi di corrente relativi ad ogni stato, si ottiene il modello completo del consumo di potenza. Una misura indiretta della potenza è preferibile principalmente a causa della difficoltà di misurare potenze per stati operativi di breve durata.

---

<sup>1</sup>Rapida variazione di tensione, più precisamente un picco di tensione di breve durata.

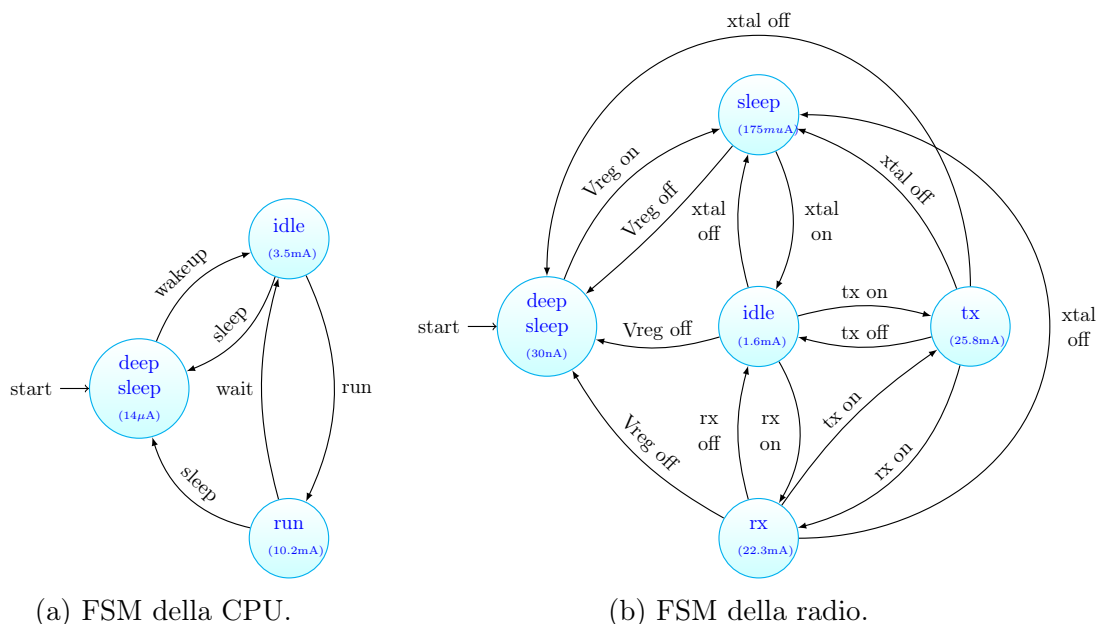


Figura 6.0.1: Stati operativi della CPU e della radio.

La figura 6.0.2 mostra il consumo di potenza profilato per il nodo referente su cui viene eseguito null'altro che il protocollo di sincronizzazione. Inoltre per semplificare il modello e rendere le misure ripetibili è stata disabilitata la re-sincronizzazione periodica del VHT. All'inizio di ogni periodo di sincronizzazione l'interrupt del timer risveglia il processore dallo stato di *deep sleep*. La CPU avvia il PLL ed entra nello *run*. Successivamente viene eseguita l'unica re-sincronizzazione del VHT e viene avviata la procedura per portare il transceiver radio dallo stato di *deep sleep* a quello di *sleep*. Dopo un attesa di  $100\mu s$ , necessaria per rendere l'alimentazione del transceiver stabile, la CPU passa nuovamente nello stato di *run* ed invia al transceiver il comando per avviare il suo oscillatore al quarzo, dopo quest'operazione la cpu torna nello stato di *idle* in attesa che la radio gli comunichi con un interrupt l'avvenuta transizione nello stato di *idle*. All'arrivo dell'interrupt la CPU passa nello stato di *run*, inizializza i registri della radio, scrive nel buffer della stessa il pacchetto da trasmettere e imposta il registro virtuale di capture del VHT che emetterà il segnale di *trigger* per l'invio del bsp. A questo punto sia radio che CPU restano nello stato di *idle*, in attesa del *trigger*. Questo tempo di attesa ha la funzione di assorbire i jitter del PLL e o eventuali altri jitter hardware. Quando il VHT emette il *trigger*, il transceiver si porta nello stato *TX* ed invia il pacchetto, mentre la CPU resta in attesa dell'interrupt di SFD o del timeout interrupt impostato nel caso per qualche ragione non si ricevesse l'SFD. La CPU all'arrivo dell'interrupt dell'SFD, abbassa il segnale di interrupt del transceiver, setta un nuovo timeout interrupt per la fine trasmissione pacchetto e si mette in

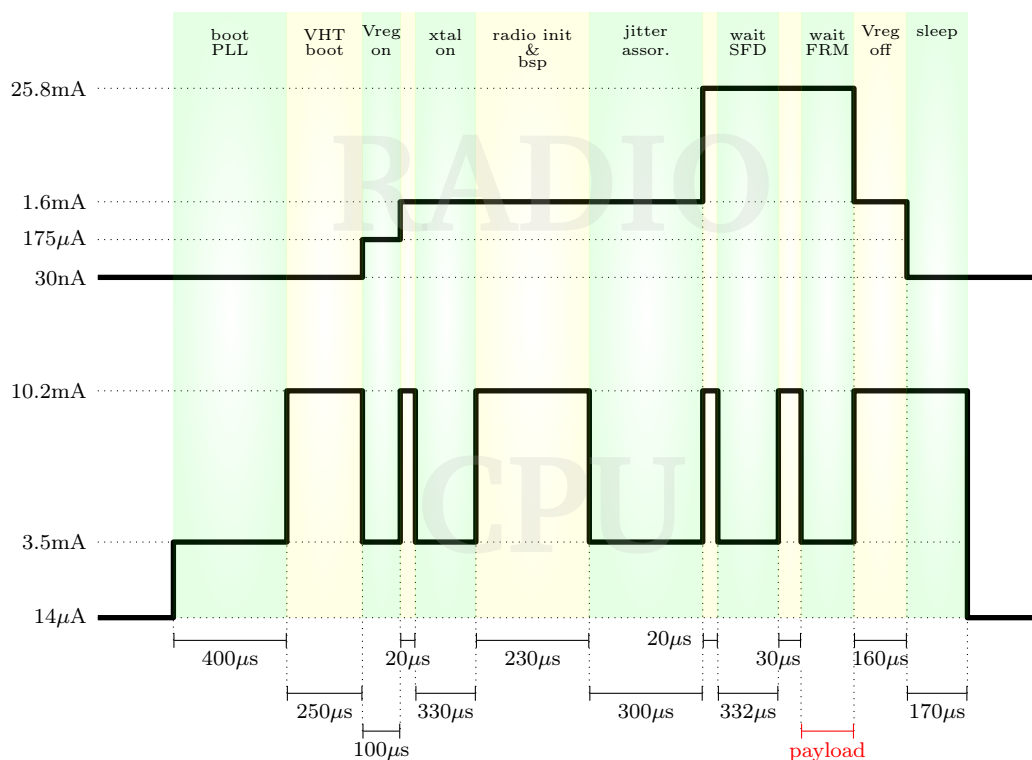


Figura 6.0.2: Consumo di corrente tracciato per il nodo reference (non in scala), radio (in alto) e CPU (in basso).

attesa di quest'ultimo. In corrispondenza dell'interrupt di fine trasmissione pacchetto (FRM\_done) il transceiver passa nello stato di *idle* mentre la CPU passa in quello di *run* per inviare al transceiver il comando di *deep sleep*, eseguire le operazioni necessarie per impostare l'invio del prossimo bsp e porre se stesso in *deep sleep*, in attesa della prossima sincronizzazione.

La figura 6.0.3 viceversa, mostra il profilo di consumo di potenza per il generico nodo (nodo non-reference). La sequenza di attivazione della CPU e del transceiver è identica a quella del nodo reference. Dopo l'intervallo di tempo necessario per assorbire i jitter hardware, la CPU invia il comando per porre in ricezione la radio, imposta un timeout interrupt a  $rx\_turnaround\_time + 2w + SFD\_trasmission\_time$ , e torna in *sleep*. Prima che il transceiver sia davvero in grado di ricevere un frame è necessario attendere l' $rx\_turnaround\_time$  (tempo necessario per calibrare la frequenza di ricezione della radio) più un eventuale tempo per assorbire eventuali jitter hardware. A metà della finestra di sincronizzazione normalizzata con l' $SFD\_trasmission\_time$ , il transceiver invia l'interrupt di SFD. La CPU all'arrivo dell'interrupt dell'SFD, abbassa il segnale di interrupt del transceiver, setta un nuovo timeout interrupt per il FRM\_done e si pone in attesa di quest'ultimo. In corrispondenza dell'interrupt di fine ricezione pacchetto il transceiver passa nello

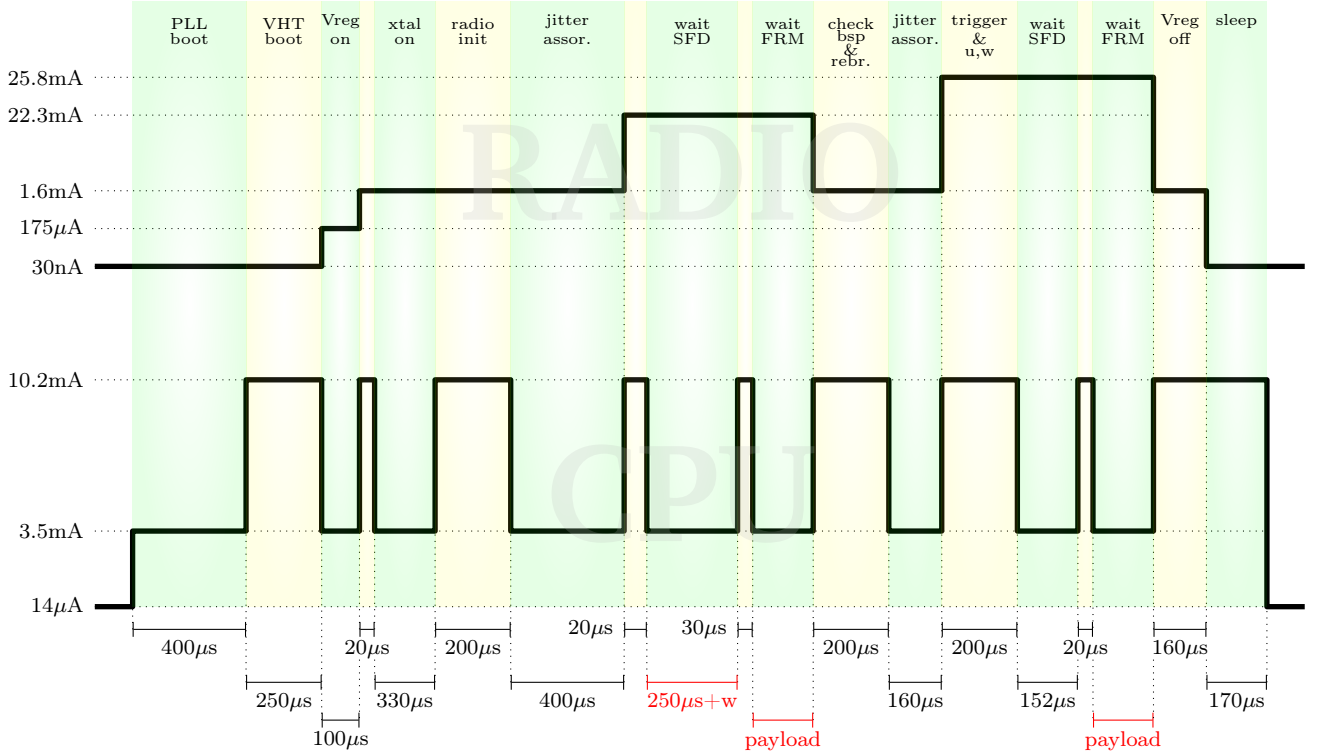


Figura 6.0.3: Consumo di corrente tracciato per il nodo non-reference (non in scala), radio (in alto) e CPU (in basso).

stato di *idle* mentre la CPU in quello di *run* ed immediatamente esegue le istruzioni per la ritrasmissione del bsp. Mentre il pacchetto viene trasmesso viene eseguito il codice del controllore FLOPSYNC per il calcolo di  $u$  e  $w$ . Non appena termina l'invio del pacchetto, CPU e transceiver entrano nello stato di *deep sleep*.

Il consumo di potenza medio di FLOPSYNC si ottiene integrando l'area sotto il profilo di corrente a meno del consumo in *deep sleep* di radio e della CPU, e poi diviso per il periodo di sincronizzazione  $T$ . Per tanto risulta che il consumo di corrente per il nodo reference e per quello da sincronizzare è:

$$I_{ref} = \frac{25.6\mu C + payload\_bytes \cdot 0.94\mu C}{T} \quad (6.0.1)$$

$$I_{sync} = \frac{37.8\mu C + payload\_bytes \cdot 1.76\mu C + w \cdot 25.79mA}{T} \quad (6.0.2)$$

parametriche sul periodo di sincronizzazione, numero di byte del payload e tempo medio della finestra  $w$ .

Con un payload di un byte (cioè senza fare piggybacking) ed il tempo medio di  $w$  preso dall'esperimento di 7 giorni della sezione 5.2.2, il consumo di potenza di FLOPSYNC è  $442nA$  per il nodo reference e  $669nA$  per il nodo generico da

sincronizzare.

La disponibilità del modello potenza permette anche di valutare l'efficacia delle ottimizzazioni di potenza impiegate in FLOPSYNC, ossia la mancanza di necessità di inviare timestamp e l'adattamento della finestra di ricezione. Ripetendo lo stesso calcolo utilizzando un pacchetto di 9 byte (8 byte di timestamp a 64 bit più un byte per il conteggio degli hop) e la finestra di ricezione fissa a  $3ms$  si ha un consumo di potenza pari a del  $567nA$  per il nodo reference e  $2.184\mu A$  per quello generico. Questo dimostra come queste ottimizzazioni siano la chiave per ottenere un consumo inferiore ai  $\mu A$ .





# Capitolo 7

## Conclusioni e punti aperti

In questo elaborato di tesi è stato presentato un nuovo protocollo di sincronizzazione per reti di sensori wireless costituito da un sistema di controllo decentralizzato, lineare, tempo invariante a tempo discreto. Lo schema, detto FLOPSYNC, fornisce una rappresentazione del tempo monotona ad ogni nodo della rete di sensori ed è capace di compensare variazioni del clock causate dalla temperatura senza la necessità di misurarla. Il protocollo non richiede la trasmissione di timestamp eccetto che nella fase di boot o desincronizzazione che si presume avvengano di rado. Essendo basato sulla teoria del controllo ed operando in maniera decentralizzata, i parametri del controllore possono essere configurati in base alle esigenze del nodo ed a quelle che saranno le sue condizioni operative, prima della messa in opera del nodo stesso. Grazie a queste caratteristiche FLOPSYNC fornisce una sincronizzazione del clock con compensazione di skew/drift, accuratezza inferiore al  $\mu s$  ed un consumo di potenza dell'ordine di qualche  $\mu A$ . Inoltre lo schema può essere efficientemente implementato usando solo operazioni in aritmetica intera con un'occupazione di RAM inferiore se comparato con quegli schemi che fanno compensazione utilizzando la regressione lineare piuttosto che compensazione di temperatura attraverso la costruzione della caratteristica temperatura-frequenza del quarzo misurata.

Da ultimo, dalle verifiche sperimentali è emerso che, opportuni accorgimenti implementativi uniti all'utilizzo di uno schema di flooding efficiente consentono di rendere la varianza dell'errore di sincronizzazione assai poco dipendente dal numero di hop.

FLOPSYNC è stato implementato e testato su una rete di sensori fatta con CO-TS. Il VHT e parte dello stack radio sono stati implementati e sono ora disponibili con il kernel Miosix.

Lavori futuri sono indirizzati nel rifinire il protocollo. Uno di questi è sicu-

ramente lo studio di una politica più intelligente di gestione della finestra di sincronizzazione che eviti di perdere la sincronizzazione anche in presenza di brusche variazioni di temperatura. Si ricorda che in ottica power è dannoso perdere la sincronizzazione in quanto successivamente si è costretti a tener accesa la radio per periodi lunghi.

Sebbene i livelli più bassi dello stack siano stati integrati in Miosix, per impieghi reali futuri si inizia a sentire la necessità di avere uno stack radio completo.

In questo lavoro si è proposto un tool per configurare FLOPSYNC che privilegiasse la semplicità e l'efficienza, fin troppo conservativo. Un interessante studio in questa direzione potrebbe essere quello di progettare un qualche meccanismo posizionato sul nodo che, attraverso il monitoraggio delle prestazioni, agisca sul controllo operando un'ottimizzazione multi-obiettivo secondo quelli che sono i vincoli imposti. Tale meccanismo è pensato per operare su scala temporale piuttosto lunga evitando così di inficiare le prestazioni complessive del nodo.

# Bibliografia

- [1] *std. 802.15.4 - 2003: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std. [Online]. Available: <http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf>
- [2] S. N. Gelyan, A. N. Eghbali, L. Roustapoor, S. A. Y. F. Abadi, and M. Dehghan, "SLTP: Scalable Lightweight Time Synchronization Protocol for Wireless Sensor Network," *Mobile Ad-Hoc and Sensor Networks*, vol. 4864, pp. 536–547, Dicembre 2007.
- [3] F. M. K. Romer, "The design space of wireless sensor networks," in *Wireless Communications, IEEE*, IEEE, Ed., vol. 11. IEEE, Dicembre 2004, pp. 54–61.
- [4] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, pp. 281–323, Maggio 2005.
- [5] D. L. Mills, "Network Time Protocol (version 3) Specification, Implementation and Analysis," Tech. Rep., Marzo 1992. [Online]. Available: <https://tools.ietf.org/html/rfc1305>
- [6] D. Mills, "Wireless Sensor Network Time Synchronization Algorithm Based on SFD," *Communications in Computer and Information Science*, vol. 334, pp. 393–400, 2013.
- [7] S. E. Khediri, N. Nasri, M. Samet, A. Wei, and A. Kachouri, "Analysis study of time synchronization protocols in wireless sensor networks," *Communications in Computer and Information Science*, vol. 3, no. 3, pp. 155–165, 2012.
- [8] D. L. Mills, U. Delaware, and J. Burbank, "Network Time Protocol Version 4: Protocol and Algorithms Specification," Tech. Rep., Giugno 2010. [Online]. Available: <https://tools.ietf.org/html/rfc5905>

- [9] “National Institute of Standards and Technology,” 2014. [Online]. Available: <http://www.nist.gov/el/isd/ieee/ieee1588.cfm>
- [10] J. Elson, L. Girod, and D. Estrin, “Fine-Grained Network Time Synchronization using Reference Broadcasts,” *SIGOPS Operating Systems Review OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, vol. 36, pp. 147–163, 2002.
- [11] S. Ganeriwal, R. Kumar, and M. B. Srivastava, “Timing-sync Protocol for Sensor Networks,” *SenSys '03 Proceedings of the 1st international conference on Embedded networked sensor systems*, vol. 36, pp. 138–149, 2003.
- [12] S. Ping, “Delay Measurement Time Synchronization for Wireless Sensor Networks,” *Intel Research*, Giugno 2003.
- [13] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, “The Flooding Time Synchronization Protocol,” *SenSys '04 Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 39–49, 2004.
- [14] J. van Greunen and J. Rabaey, “Lightweight Time Synchronization for Sensor Networks,” *WSNA '03 Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pp. 11–19, 2003.
- [15] M. L. Sichitiu and C. Veerarittiphan, “Simple, Accurate Time Synchronization for Wireless Sensor Networks,” *Wireless Communications and Networking*, vol. 2, pp. 1266–1273, Marzo 2003.
- [16] S. Rahamatkar<sup>1</sup> and D. A. Agarwal, “A Reference based, Tree Structure Time Synchronization approach and its Analysis in WSN,” *International Journal of Ad hoc, Sensor & Ubiquitous Computing (IJASUC)*, vol. 2, no. 1, pp. 1266–1273, Marzo 2011.
- [17] J. Chen, Q. Yu, Y. Zhang, H.-H. Chen, Fellow, and Y. Sun, “Feedback-Based Clock Synchronization in Wireless Sensor Networks: A Control Theoretic Approach,” *IEEE Transactions on vehicular technology*, vol. 56, no. 6, pp. 1963–1973, Luglio 2010.
- [18] K.-L. Noh, Q. M. Chaudhari, E. Serpedin, and B. W. Suter, “Novel Clock Phase Offset and Skew Estimation Using Two-Way Timing Message Exchanges for Wireless Sensor Networks,” *Communications, IEEE Transactions on*, vol. 55, no. 4, pp. 766–777, Aprile 2007.

- [19] K.-L. Noh, E. Serpedin, and K. Qaraqe, “A New Approach for Time Synchronization in Wireless Sensor Networks: Pairwise Broadcast Synchronization,” *Communications, IEEE Transactions on*, vol. 7, no. 9, pp. 3318–3322, Settembre 2008.
- [20] A. Leva, M. Maggio, A. V. Papadopoulos, and F. Terraneo, *Control-Based Operating System Design*. London, UK: Institution of Engineering and Technology, 2013.
- [21] M. Nakazawa, Y. Nakamura, and S. Miyashita, “Frequency-temperature characteristics of quartz crystal flexure bars and quartz crystal tuning forks,” *IEEE Transactions on Sonics and Ultrasonics*, vol. 26, no. 5, pp. 369–376, 1979.
- [22] P. Marchetto, A. Strickhart, R. Mack, and H. Cheyne, “Temperature compensation of a quartz tuning-fork clock crystal via post-processing,” in *IEEE International Frequency Control Symposium (FCS)*, 2012, pp. 1–4.
- [23] T. Schmid, P. Dutta, and M. B. Srivastava, “High-Resolution, Low-Power Time Synchronization an Oxymoron No Mmore,” *IPSN '10 Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 151–161, 2010.
- [24] A. Leva, M. Maggio, A. V. Papadopoulos, and F. Terraneo, *Control-Based Operating System Design*. London, UK: Institution of Engineering and Technology, 2013, ch. 2, pp. 11–30.
- [25] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, “Efficient Network Flooding and Time Synchronization with Glossy,” *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, pp. 73–84, 2011.
- [26] “Autonomous sensing and communication in a cubic millimeter.” [Online]. Available: <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>
- [27] “Crossbow technology.” [Online]. Available: <http://www.xbow.com>
- [28] “Moteiv corporation.” [Online]. Available: <http://www.moteiv.com>
- [29] “Intel.” [Online]. Available: [www.intel.com/](http://www.intel.com/)
- [30] “Ember corporation.” [Online]. Available: <http://www.ember.com>

- [31] “Tinyos.” [Online]. Available: <http://www.tinyos.net/>
- [32] “Contiky.” [Online]. Available: <http://www.contiki-os.org/>
- [33] “Zigbee alliance.” [Online]. Available: <https://www.zigbee.org/>
- [34] *std. 802.15.4 - 2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANs)*, IEEE Std. [Online]. Available: <http://standards.ieee.org/getieee802/download/802.15.4-2006.pdf>
- [35] “Stm32vldiscovery.” [Online]. Available: <http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/PF250863>
- [36] “Cc2520.” [Online]. Available: <http://www.ti.com/product/cc2520>
- [37] “Miosix.” [Online]. Available: <http://miosix.org/index.html>
- [38] “Metodi per il calcolo della radice quadrata.” [Online]. Available: [http://it.wikipedia.org/wiki/Metodi\\_per\\_il\\_calcolo\\_della\\_radice\\_quadrata](http://it.wikipedia.org/wiki/Metodi_per_il_calcolo_della_radice_quadrata)