La sincronizzazione dei clock
parte 1

Corso di reti
per l'automazione industriale

Prof. Orazio Mirabella

Sincronizzazione

Un sistema distribuito è costituito da un insieme di processori che comunicano attraverso lo scambio di messaggi per realizzare una applicazione. Solitamente questi processori operano in modo autonomo (seppur coordinato) e non hanno accesso ad un clock comune in quanto non necessitano di un riferimento temporale unico.

Lo scopo di sincronizzare due computer è assicurare che diversi processi fisicamente distanti abbiano la stessa nozione del tempo.

Perché la sincronizzazione è necessaria

- Applicazioni per controllo di processo
- Applicazioni di misura
- Applicazioni transazionali
- Applicazioni relative alla gestione di protocolli di comunicazione (MAC)
- Molte applicazioni nelle WSNs necessitano eventi con time stamps.
- Un evento fisico è rivelato da diversi nodi (es. tracking)
- Quando nelle WSNs è adottata una politica "Energy Save" i nodi devono essere sincronizzati in modo da essere contemporaneamente in *Idle* or *Active* mode.
- L'ordine dei messaggi può cambiare durante la trasmissione

Quindi

In sintesi la sincronizzazione è utile :

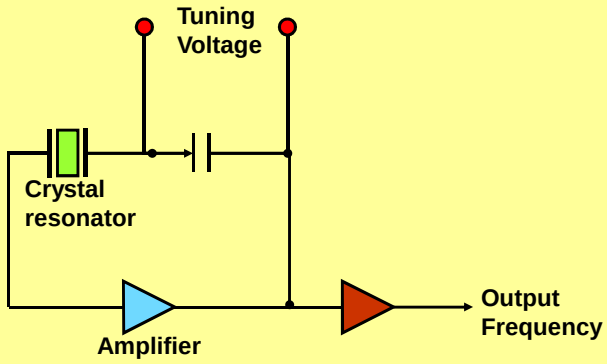
- Per Stabilire **l'istante** in cui un evento si è vericato
- **L'intervallo** di tempo fra due eventi
- **L'ordine relativo** fra vari eventi.

Tipi di Clock

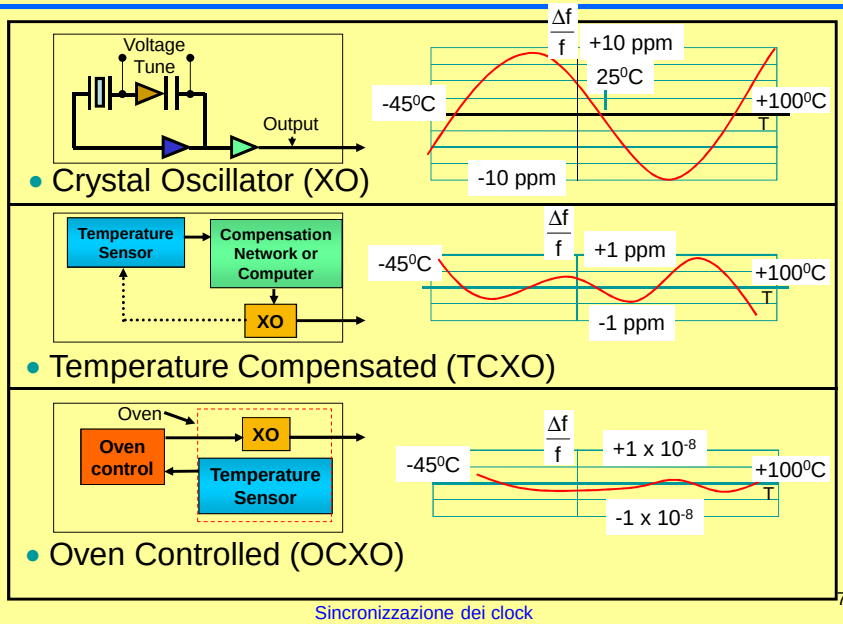
- Digital Clock
  - E' realizzato mediante un contatore digitale.
  - E' sensibile a variazioni della tensione di alimentazione o della temperatura.
  - La frequenza è fissa o con variazioni limitate (oscillatori al quarzo)
- Software Clock
  - Il valore di un contatore è convertito in un tempo.
  - Si può aggiungere/sottrarre un tempo di offset per ottenere la sincronizzazione.

Oscillatore al quarzo

L'elemento base per la generazione del clock è un oscillatore al quarzo



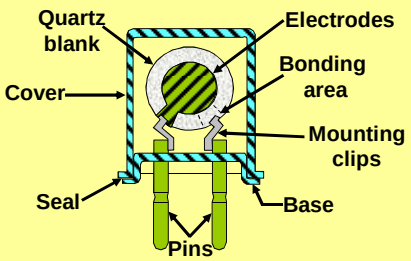
Stabilizzazione di un oscillatore al quarzo

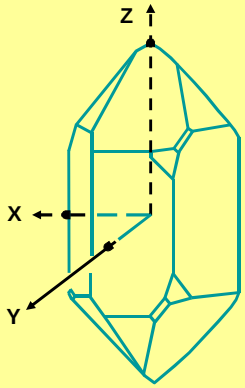


Il cristallo di quarzo

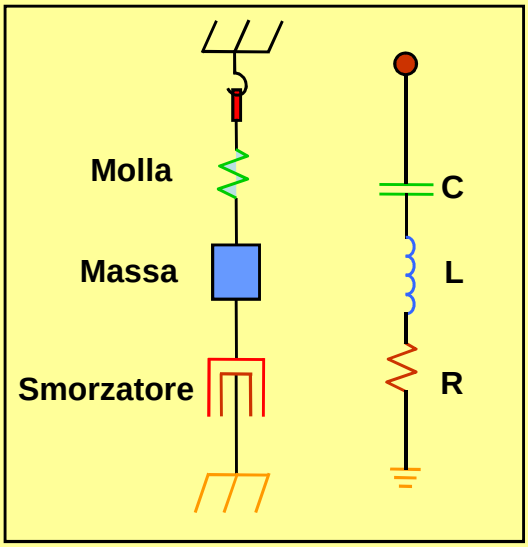
- L'effetto piezoelettrico di una lamina di quarzo lo rende equivalente ad un oscillatore meccanico.

Modo flessione Modo estensione





Circuito Equivalente

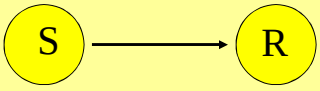


Principi base per la Sincronizzazione

- La sincronizzazione richiede la comunicazione.

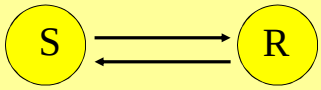
Non esiste sincronizzazione senza comunicazione

- Il sender periodicamente invia un messaggio contenente il suo "Clock corrente" (Timestamp).
- Il ricevitore si sincronizza col Sender modificando il suo clock al valore del timestamp del messaggio ricevuto (funziona se la latenza è piccola rispetto alla precisione desiderata)



Principi base per la Sincronizzazione

- Il Sender calcola l'errore di fase misurando il round trip-time totale, inviando un messaggio e ricevendo la risposta dal receiver (Funziona anche se la latenza è elevata rispetto all'accuratezza desiderata)



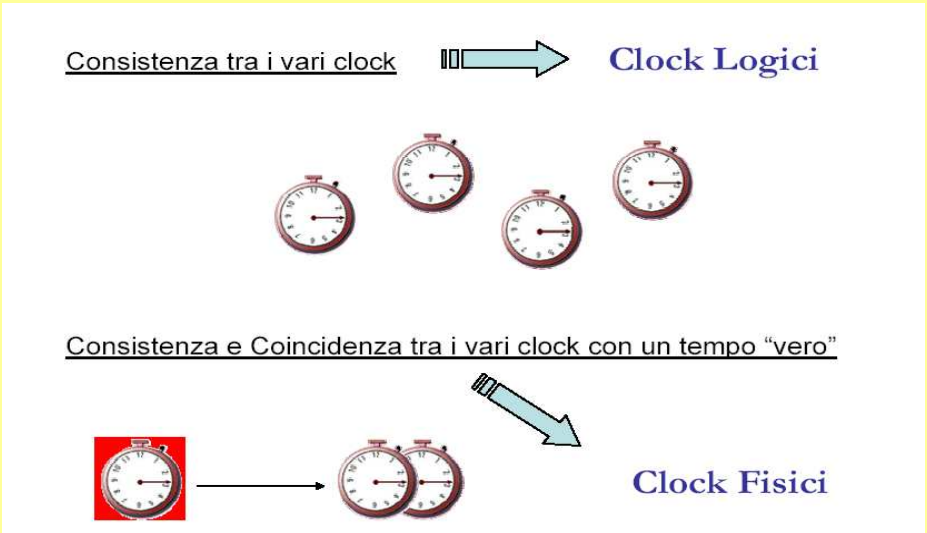
Problemi nella sincronizzazione

- La progettazione di algoritmi per la sincronizzazione di clock presenta diversi problemi:
- A causa dei diversi ritardi di trasmissione ogni processo non può avere una vista istantanea globale di tutti gli altri clock.
- anche se tutti i *clock* fossero inizializzati contemporaneamente, essi non rimarrebbero sincronizzati a causa del *drift* nei rispettivi periodi
- la presenza di elementi malfunzionanti (faulty) distribuiti nel sistema, complica ulteriormente il raggiungimento di una visione unica del tempo

Il clock logico di Lamport (1/9)

- Lamport ha evidenziato che la sincronizzazione dei clock non deve per forza essere assoluta: se due processi non interagiscono, non occorre che i loro clock siano effettivamente sincronizzati.
- Inoltre in genere non ha importanza che i processi si accordino sul tempo esatto, ma che si accordino sull'ordinamento temporale degli eventi.
- In molti casi ha importanza che ci sia consistenza interna dei vari clock nella rete: essere d'accordo su un tempo comune, senza preoccuparsi se questo sia effettivamente il tempo "vero". Si parla allora di clock logici.

Il clock logico di Lamport (2/9)



Il clock logico di Lamport (3/9)

- Per sincronizzare i clock logici, Lamport ha definito la relazione "accade prima":

a->b significa "a" accade prima di "b"

- cioè tutti i processi di tutte le stazioni sono d'accordo sul fatto che "a" accade prima di "b".
- Valgono le seguenti proposizioni:
- 1. Se "a" e "b" sono eventi di uno stesso processo ed "a" viene prima di "b" allora a->b è vera;
- 2. Se "a" è l'evento spedizione messaggio da parte di in processo e "b" è l'evento ricezione dello stesso messaggio ; da parte di un altro processo, allora a->b è vera.
- 3. Se valgono a->b e b->c allora vale anche a->c

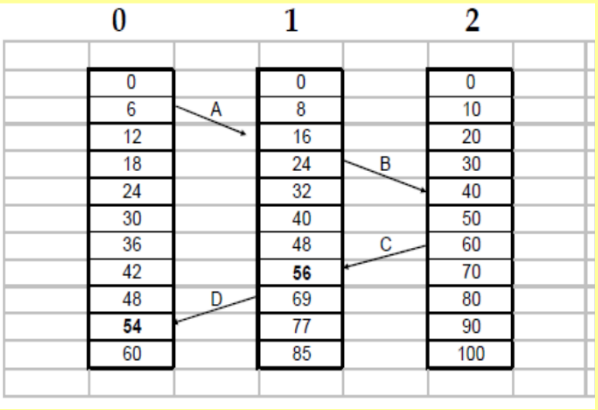
Il clock logico di Lamport (4/9)

- Assegniamo un valore del "clock" a ciascun evento a e b
  - se **a→b allora clock(a) < clock(b)**
    Poichè il tempo non può andare all'indietro
- Se *a* e *b* si verificano su processi differenti che non scambiano messaggi, allora nè a → b nè b→a sono vere.
  - Questi eventi sono concorrenti

Il clock logico di Lamport (5/9)

Se si devono apportare correzioni al clock, queste vanno fatte sempre in avanti.

Per assegnare un tempo agli eventi, ciascun messaggio contiene il suo tempo di partenza,come rilevato dal clock del processo mittente.



Nella interazione fra processi remoti occorre tenere conto del valore dei rispettivi clock.

Nella figura, i tre processi 0,1,2 hanno clock con diversa velocità

(ogni sender applica il proprio Time-Stamp al messaggio inviato).

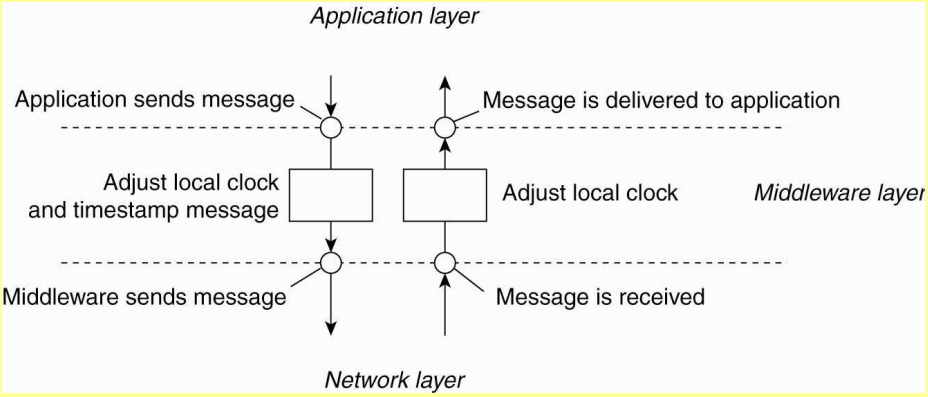
I messaggi C e D arrivano a destinazione prima del loro tempo di trasmissione

Il clock logico di Lamport (6/9)

- La soluzione di Lamport a questo problema di sincronizzazione discende dalla relazione accade-prima (happened-before): dal momento che il messaggio C parte al tempo 60, deve arrivare ad un tempo 61 o superiore.
- Poichè ciascun messaggio contiene il suo tempo di partenza, quando il messaggio arriva a destinazione, se il ricevente ha un clock che segna un tempo che precede quello della spedizione del messaggio, egli può spostare in avanti il proprio clock in modo che segni un tempo pari al tempo di trasmissione del messaggio più uno.
- In tale modo viene garantita la coerenza fra i vari clock.

Il clock logico di Lamport (7/9)

Lo scambio dei messaggi fra i vari nodi richiede che un midlleware in ogni nodo si occupi di adattare il valore dei clock per renderli coerenti.

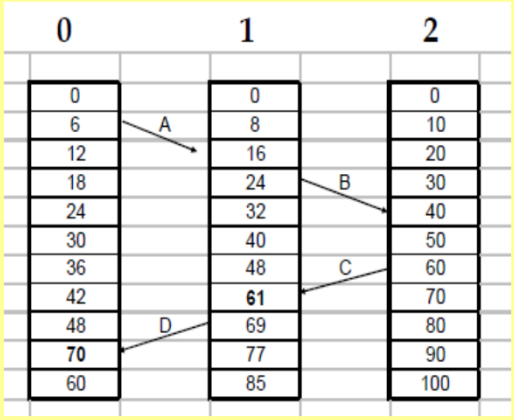


Il clock logico di Lamport (8/9)

- Quindi il clock logico può essere sincronizzato col seguente approccio:
- Sia C_i il valore del clock per il processo P_i
- Prima di eseguire un evento P_i esegue l'operazione $C_i \leftarrow C_i + 1$.
- Quando il processo P_i invia un messaggio m a P_j esso pone il timestamp $ts(m)$ uguale a C_i dopo aver eseguito l'operazione precedente.
- Appena ricevuto il messaggio m, il processo P_j aggiusta il suo contatore locale al valore $C_j \leftarrow \max\{C_j, ts(m)\}$, dopo di che esegue il primo step e consegna il messaggio all'applicazione.

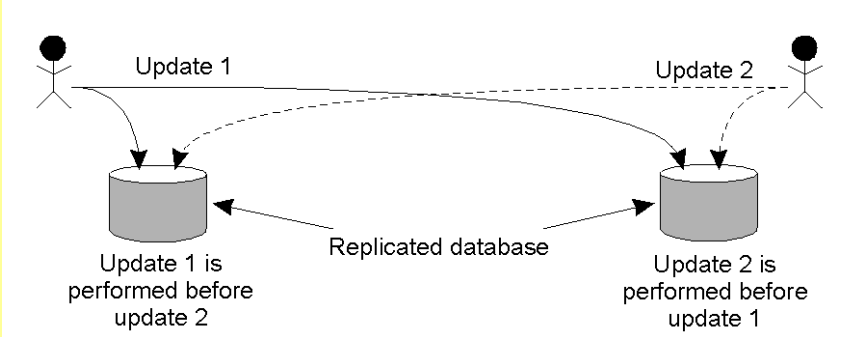
Il clock logico di Lamport (9/9)

Questo algoritmo permette di stabilire un ordinamento temporale degli eventi che hanno luogo in un sistema distribuito;



I tempi assegnati agli eventi possono non essere corrispondenti ai tempi **"veri"** in cui gli eventi si sono verificati ma garantiscono un corretto ordine temporale degli eventi.

Un Problema di sincronizzazione complesso



- Si vuole aggiornare un database replicato:
- - Update1 somma 100 euro ad un account.
- - Update2 calcola e somma 1% di interesse allo stesso
- A causa del ritardo della rete, l'aggiornamento non avviene nell'ordine corretto nelle due copie del database. !!!!!!!!!!
- Occorre garantire che i due database contengano lo stesso valore e che le due operazioni siano realizzate nello stesso ordine.

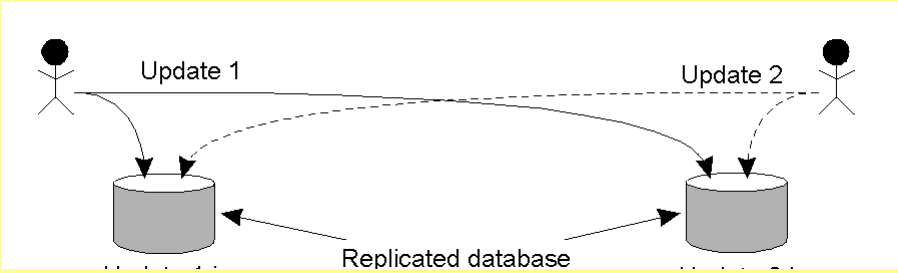
Soluzione: Totally-Ordered Multicasting (1/3)

- Un messaggio multicast viene spedito a tutti i processi del gruppo, incluso il sender, insieme al timestamp del sender.
- In ciascun processo, il messaggio ricevuto è aggiunto ad una coda locale, ordinato secondo il timestamp.
- Dopo aver ricevuto un messaggio, un timestamp di ack è spedito, a tutto il gruppo.
- Poichè vale la relazione "happens-before" il timestamp contenuto nell'ack è sempre maggiore di quello del messaggio originale.

Totally Ordered Multicasting (2/3)

- Solo quando un messaggio è marcato come "confermato" da tutti gli altri processi, esso verrà rimosso dalla coda e consegnato all'applicazione in attesa.
- I clock di Lamport garantiscono che ciascun messaggio ha un unico timestamp, e conseguentemente, che tutte le code locali di ciascun processo hanno lo stesso contenuto.
- In questo modo, tutti i messaggi sono consegnati e/o processati nello stesso ordine dovunque, e gli aggiornamenti possono avvenire in maniera consistente.

Totally-Ordered Multicasting (3/3)



- Update 1 è time-stamped and multicast. Viene sommato alle code locali.
- Update 2 è time-stamped and multicast. Viene sommato alle code locali.

- L'Ack per Update 2 sent/received. Update 2 può ora essere processato.
- L'Ack per Update 1 sent/received. Update 1 può ora essere processato.

(Nota: tutte le code sono uguali poichè l'uso dei timestamps garantisce che la relazione "happens-before" è verificata, poichè i messaggi sono ordinati in base al loro timestamp)

Sincronizzazione dei clock fisici (1/4)

- Spesso non basta un ordinamento temporale non ambiguo, ma ha importanza il tempo vero in cui si verificano gli eventi. Per questi sistemi non è sufficiente il clock logico ma bisogna fare riferimento ai clock fisici.

- Per ragioni di efficienza e ridondanza è auspicabile che nel sistema ci siano diversi clock fisici;
- ne conseguono due problemi:
  - come sincronizzarli col tempo del mondo reale;
  - come sincronizzarli tra loro;

Sincronizzazione dei clock fisici (2/4)

- 50 laboratori sparsi nel mondo che hanno orologi atomici basati su cesio 133
- Periodicamente ogni laboratorio comunica al BIH(Bureau International de l'Heure di Parigi) quanti tick ha battuto il proprio orologio
- BIH esegue la media di questi valori per produrre il TAI, Tempo Atomico Universale
- Il giorno del TAI dura circa 3 microsecondi in meno del giorno solare
- Il BIH risolve il problema inserendo del tempo di compensazione

Sincronizzazione dei clock fisici (3/4)

- Il sistema TAI compensato, prende il nome di UTC, Universal Coordinated Time, che è l'attuale standard internazionale.
- Per mettere a disposizione l'UTC a quegli utenti che necessitano di un tempo preciso, esiste una stazione radio a onde corte detta WWV che emette un beep ad ogni secondo UTC.
- Per poter compensare i ritardi introdotti dalla propagazione del segnale nell'atmosfera bisogna conoscere con accuratezza la distanza tra trasmettitore e ricevitore

Sincronizzazione dei clock fisici (4/4)

Dati due processi c e c', essi si diranno **sincronizzati** al più di δ al tempo T se:

$$|c(T) - c'(T)| < \delta$$

cioè se i due processi raggiungeranno il valore del tempo T al più con δ secondi di ritardo l'uno dall'altro.

Definiamo **time server process** il processo che si occupa della sincronizzazione. Ciascun time server process può leggere il **clock hardware (HC)** dell'host al tempo t.

Il **clock logico (LC)** è implementato via software come:

$$LC(t) = HC(t) + A(t)$$

dove **A(t)** è la **funzione di correzione** dipendente dal tempo.

I modelli di fallimento

- La presenza di fault (guasti) nel sistema diminuisce l'efficienza dei vari algoritmi di sincronizzazione.
- Si definisce fallimento, la deviazione del sistema da un comportamento corretto a causa di qualche malfunzionamento.
- Tutti i componenti del sistema (i clocks, i processori dei singoli nodi ed i link di comunicazione) possono essere soggetti a guasti/malfunzionamenti.

Clock fault

- Con riferimento ai clock, possono essere considerati guasti di tipo incontrollato (di tipo bizantino) che possono produrre valori molto inaccurati e/o contrastanti l'un l'altro, oppure fallimenti di temporizzazione più limitati, (che escludono comportamenti maliziosi) e vengono definiti *Timing failure.*
- *Clock timing failure*: il clock ha un funzionamento scorretto nel senso che non è ρ- bounded.
- *Clock byzantine failure*: il clock fornisce informazioni inaccurate, scorrette o in conflitto. Questo tipo di guasto include la presenza dei cosidetti dual-faced clocks (clock a doppia faccia) i quali forniscono differenti valori del clock a processori differenti.

Processor fault

Vanno considerate tre possibilità:

- Processor crash failure: il processore ad un certo istante cessa per sempre la sua attività. Questo guasto è facile da individuare, e si recupera ridondando i processori.
- Processor performance failure: il processore completa le proprie computazioni in un tempo superiore a quello impiegato solitamente. Questo tipo di guasto non può essere rilevato dagli altri elementi del sistema che continuano ad operare in modo normale.
- Processor Byzantine failure: il processore esegue delle computazioni incontrollate. Questo tipo di guasto è il più dannoso in quanto, ad esempio un errore nel software causato da un guasto nella RAM porta ad eseguire operazioni non previste. Inoltre, è un errore che può ripetersi sporadicamente ed è difficile da individuare.

Communication fault

- *Link omission failure*: un link l da un processore pi ad un processore pj commette una *omission failure* su un messaggio **m**, se m viene inserito nel buffer d'uscita di Pi ma non è trasportato fino al buffer d'ingresso di pj. Il fallimento consiste quindi nella perdita del messaggio da trasportare
- *Link performance failure*: il link, per trasportare un messaggio m, impiega un tempo superiore a quello massimo specificato. Ovviamente, questa definizione può applicarsi solo a link basati su protocolli di comunicazione che garantiscono un ritardo massimo.
- Poiché la comunicazione di un messaggio coinvolge, oltre al link anche un processo sender ed uno receiver, una link failure può dipendere da una failure in uno (o più) elementi o in uno o più nodi del sistema.

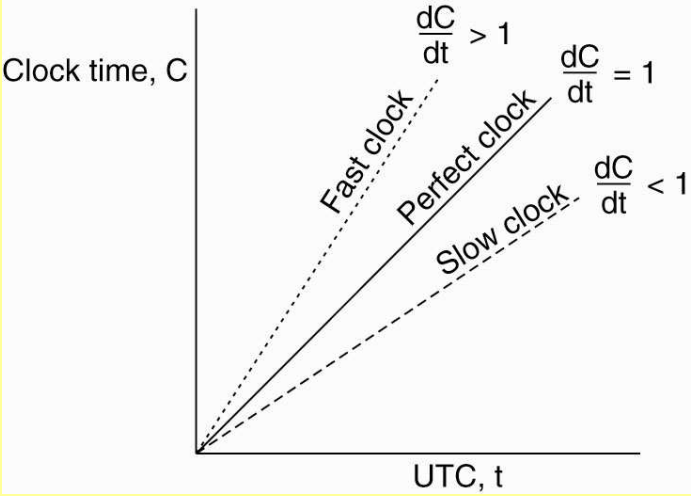
Algoritmi di sincronizzazione in presenza di Fault

- hanno la proprietà che se un numero sufficiente di processi sono non faulty allora i loro clock restano sincronizzati.
- Lamport ha sviluppato due algoritmi:
- **Interactive Convergence Algorithm (CNV);**
- **Interactive Consistency Algorithm (COM).**
- Questi algoritmi assumono che:
- **A0**: *tutti i clock siano inizialmente sincronizzati approssimativamente allo stesso valore;*
- **A1**: *il clock di un processo non faulty avanza approssimativamente con lo stesso passo del tempo reale;*
- **A2**: *un processo non faulty p può leggere la differenza Δqp tra il clock di un altro processo non faulty q ed il suo, con un errore al più di ε.*

Parametri legati al clock

- **Time:** Il tempo di un clock in una macchina p è dato dalla funzione $C_p(t)$, dove $C_p(t) = t$ per un clock perfetto.
- **Stabilità** di un clock: si intende la sua capacità di mantenere una frequenza costante;
- **Accuratezza:** si intende quanto è accettabile il suo tempo rispetto all'UTC;
- **Skew:** Lo skew di un clock è la differenza istantanea del clock rispetto al clock perfetto. Lo skew di un clock C_a rispetto ad un clock C_b al tempo t è $(C_a(t) - C_b(t))$.
- **Drift (rate):** Il drift del clock C_a è la derivata del valore del clock rispetto al tempo, indicato $C'_a(t)$. Il drift del clock C_a rispetto al clock C_b al tempo t è $(C'_a(t) - C'_b(t))$ cioè è la differenza fra le due letture per unità di tempo.

Parametri legati al clock



Relazione fra clock time and UTC quando il clock avanza a differenti velocità.

Proprietà dei clock

- Un clock hardware H è **corretto** se il drift rate è entro una soglia ρ>0:

  ciò significa che , dati due istanti t e t' :

  se **t>t'** allora

  (1-ρ) (t'-t) ≤ H(t')-H(t) ≤ (1+ρ) (t'-t).

- Un clock C è **monotono** se avanza soltanto, quindi :

  se **t>t'** allora **C(t)>C(t')**

SINCRONIZZAZIONE

- Gli errori dei clock sono tipicamente nell'ordine di 10^{-5}, che corrispondono a 10 parti per milione di clock ticks
- Il costruttore di norma fornisce il valore del *maximum drift rate* **(ρ).**
- Dalla formula **(1-ρ) (t'-t) ≤ H(t')-H(t) ≤ (1+ρ) (t'-t)** possiamo ricavare il worst case drift fra due clock pari a

  (1+ρ) (t'-t) - (1-ρ) (t'-t)

  Posto **(t'-t)= Δ*t*** ➔ **(1+ρ) Δ*t* - (1-ρ) Δ*t*** = **2ρΔ*t*.**
- Pertanto, il worst case drift δ ogni Δ*t* secondi, sarà al massimo **δ** =2ρΔ*t*.
- Per garantire che due clock non differiscano mai per più di **δ**, i clock devono essere risincronizzati ogni **Δ*t*= δ/2ρ** secondi usando uno dei vari algoritmi di sincronizzazione.

SINCRONIZZAZIONE

Gli algoritmi di sincronizzazione di clock fisici si possono classificare in:

- **Deterministici**: assumono che esista un upper bound sul ritardo di trasmissione. Garantiscono un limite massimo sulla differenza tra due clock qualsiasi.
- **Probabilistici**:non fanno alcuna assunzione sul massimo ritardo di trasmissione. Assumono note le distribuzioni dei ritardi
- **Statistici**: non fanno alcuna assunzione sul massimo ritardo di trasmissione. Assumono nota soltanto la deviazione standard dei ritardi.

SINCRONIZZAZIONE

- "**sincronizzazione di clock interna**" : non è disponibile il tempo reale, e l'obiettivo è minimizzare la massima differenza esistente tra ogni coppia di clock
- "**sincronizzazione di clock esterna**": un clock sorgente mostra il tempo reale e l'obiettivo degli altri clock è essere vicini a questo tempo sorgente il più possibile.

SINCRONIZZAZIONE

- Sia **S** la sorgente di tempo esterna UTC;
- sia **D** la **precisione**;
- sia **I** un intervallo di tempo;
- Consideriamo la presenza di N clock da sincronizzare

- la sincronizzazione esterna tiene conto della precisione rispetto ad un agente esterno:
- $| S(t)-C_i(t) | < D \qquad 1<i<N \qquad$ per ogni t in I
  I vari C_i sono accurati entro la precisione D;

- la sincronizzazione interna è concordata tra le varie entità interne:
  $| C_i(t)-C_j(t) | < D \qquad 1<i,j<N \qquad$ per ogni t in I

ALGORITMI DI SINCRONIZZAZIONE PER un SISTEMA ASINCRONO

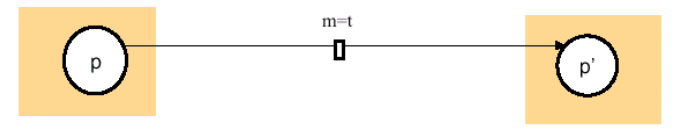
- gli algoritmi si diversificano in :

- **Algoritmi distribuiti** : **Tutti i processi cooperano, comunicando fra di loro, per risolvere un dato problema**.

- **Algoritmi centralizzati** :
  Esiste un particolare processo, chiamato coordinatore, che gestisce l'interazione fra processi.

Sincronizzazione in un sistema asincrono

- Supponiamo di avere due processi P e P';
- sia Tcom il tempo di trasmissione di un messaggio m da P a P';
- P invia un messaggio con il clock locale posto a t;
- P' riceve il messaggio e pone il proprio clock locale a t +Tcom



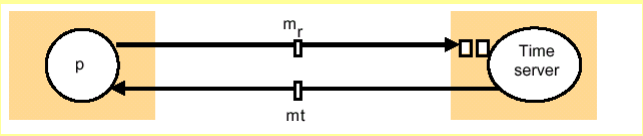
Sia Tmin la soglia minima per Tcom;
sia Tmax la soglia massima per Tcom

Se P' pone il clock a t + Tmin (oppure a t + Tmax)
allora si ha un errore $\leq \Delta = (Tmax - Tmin)$
Se P' pone il clock a t + (Tmin + T max) / 2
allora il massimo errore che si può avere sarà $\leq \Delta/2$

Algoritmo di Cristian (1/4)

E' uno dei primi algoritmi di sincronizzazione. Adatto per sistemi distribuiti asincroni :E' Centralizzato.

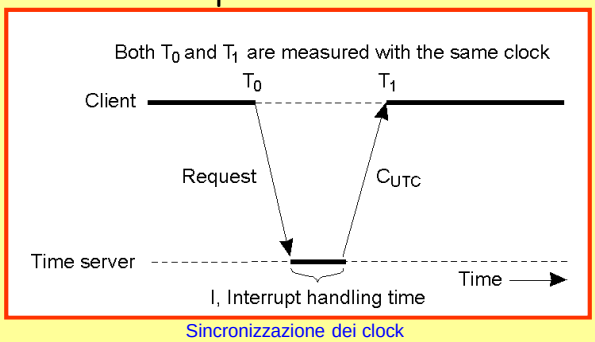
- *Assumiamo che una macchina (il time server) abbia un ricevitore WWV e tutte le altre macchine debbano essere sincronizzate con esso.*
- Ogni $\delta/2\rho$, ciascuna macchina client spedisce un messaggio al time server chiedendo il tempo corrente.
- Il Time server risponde inviando un messaggio mt contenente il tempo corrente C_{UTC}.



Algoritmo di Cristian (2/4)

Alla ricezione della risposta (all'istante T_1) il client:
- controlla il clock C_{utc}
- calcola il ritardo della rete come $T_{round} = (T_1 - T_0)$ *(round trip)*
- pone il clock a **t+Tround /2**

L'algoritmo permette la sincronizzazione solo se il tempo T round è breve ed i tempi di andata e ritorno sono uguali.



Algoritmo di Cristian (3/4)

- Problema di base: se il clock del client è più veloce del tempo reale, il valore ricevuto di C_{UTC} sarà più piccolo del tempo locale C.
- Occorrerebbe arretrare il clock locale
- Ma il clock non deve mai andare indietro
  - E allora cosa si può fare?
    - ***Occorre rallentare gradualmente il clock del client, aggiungendo meno tempo per ogni tick.***

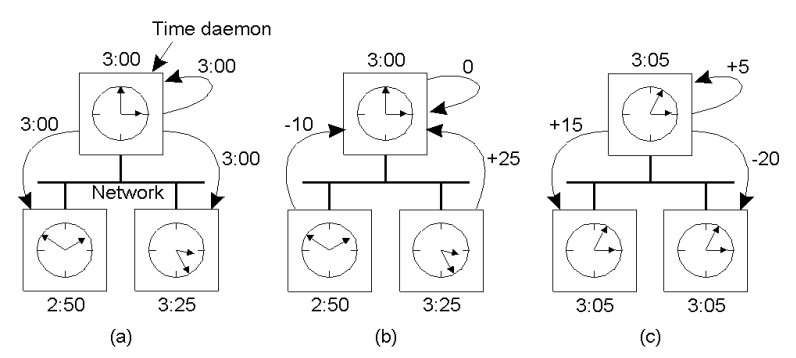
Algoritmo di Cristian (4/4)

- Ulteriore problema – il ritardo one-way dal server al client è " rilevante" e può variare notevolmente. Ciò può introdurre un notevole errore nella sincronizzazione.
  - Come l'algoritmo di Cristian risolve il problema?
    - ***Si deve misurare questo ritardo e aggiungerlo a C_{UTC}.***
    - La migliore stima del ritardo è **$(T_1 - T_0)/2$**.
  - Nel caso in cui **$T_1 - T_0$** supera una certa soglia, la misura va ignorata { outliers}
  - Si può migliorare la stima sottraendo l'interrupt handling time del server.

Algoritmo di Berkeley (1/5)

- Gusella & Zatti, 1989
- L'algoritmo di Berkeley è solitamente usato in reti dove nessuna macchina è dotata di un ricevitore WWV.
- Una stazione, che ricopre il ruolo di time server, interroga periodicamente tutte le macchine per chiedere il valore del loro tempo locale.
- Quindi calcola un tempo medio e risponde ad ogni macchina dicendogli di:
- Incrementare il clock se questo è minore del tempo medio;
- Decrementare il clock se questo è maggiore del tempo medio (decrementa la velocità, non il valore assoluto);

Algoritmo di Berkeley (2/5)

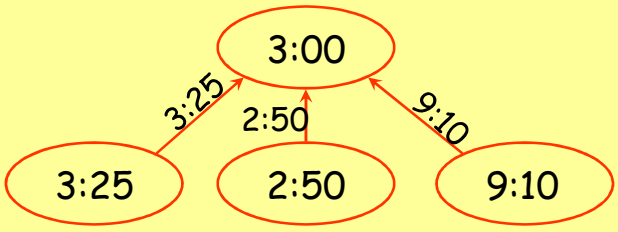


Richiede un time server attivo ed un coordinatore che

- - chiede periodicamente a tutti il valore del clock;
- - calcola una media tramite una stima;
- - indica chi deve accelerare o rallentare inviando un valore correttivo $\pm$ d .
- Si basa sul tempo T round *(round- trip)* fra client e server, che si assume limitato superiormente;

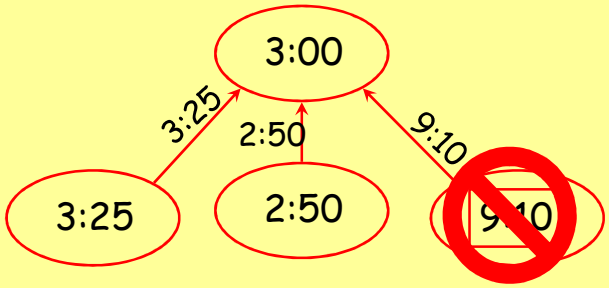
Algoritmo di Berkeley (3/5)

- La media non considera i tempi troppo lontani o con valori anomali.
- Questo è un algoritmo di tipo fault- tolerant.
- Anch'esso presenta dei problemi, legati al server centrale che comporta limiti all'affidabilità, tuttavia l'algoritmo prevede che in caso di guasto si elegga un nuovo server.
- Un altro problema è legato al caso in cui T round non è limitato : in quel periodo non è garantito il controllo della sincronizzazione.



1. Request timestamps from all slaves

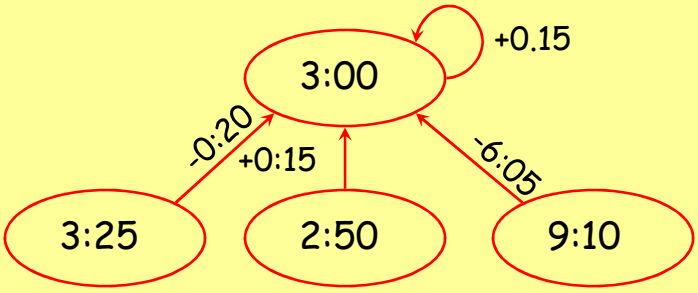
Algoritmo di Berkeley (4/5)



2. Compute fault-tolerant average:

$$\frac{3:25 + 2:50 + 3:00}{3} = 3:05$$

Algoritmo di Berkeley (5/5)



3. Send offset to each client

Network Time Protocol (NTP) (1)

- protocollo per la distribuzione del tempo su Internet

Obiettivi del progetto:

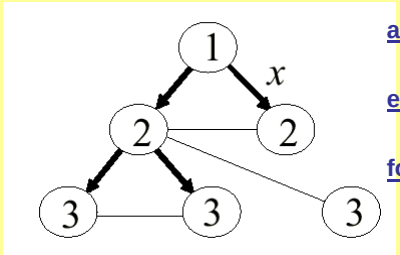
- fornire un servizio che permetta a tutti clienti della rete una sincronizzazione accurata del clock fisico con UTC
- uso di tecniche statistiche di filtro per trattare l'informazione proveniente da vari server
- servizio affidabile che tollera perdite di connessione basate su ridondanza dei server e dei cammini;
- tecniche di riconfigurazione in caso di guasto
- scalabilità: permette la sincronizzazione frequente anche in presenza di molti nodi
- protezione da interferenze casuali e/o intenzionali

Network Time Protocol (NTP) (2)

- In NTP uno o più server principali si sincronizzano direttamente a sorgenti di riferimento esterne come i clock radio.
- I server secondari si sincronizzano a questi server principali e ad altri nella sottorete di sincronizzazione.

maggiore affidabilità

minore precisione

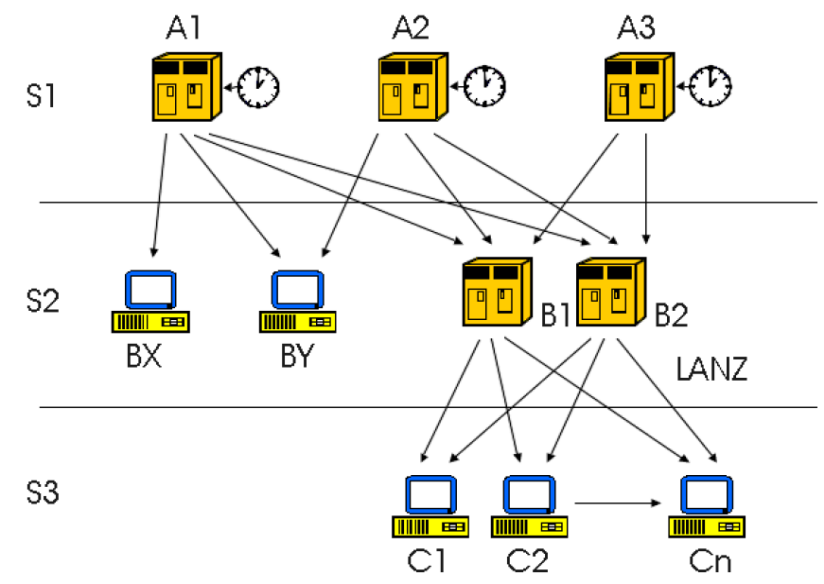


archi: controllo di sincronizzazione

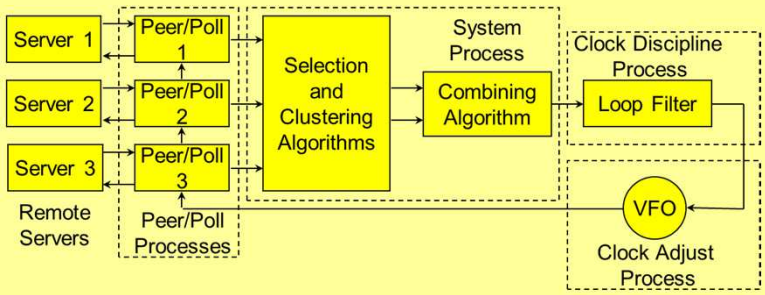
etichette di nodi: livelli

foglie: stazioni di utente

Network Time Protocol (NTP) (3)



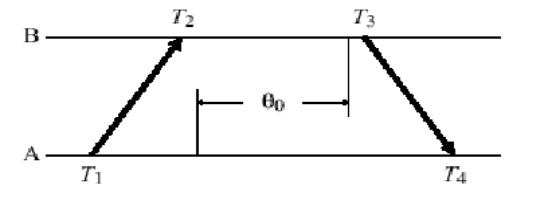
NTP (4): architettura del nodo



- Un Peer process viene eseguito ad ogni ricezione di un pacchetto.
- I Poll process spediscono pacchetti a un server ad intervalli regolari.
- Il System process viene eseguito ad ogni nuovo aggiornamento.
- Il Clock discipline process implementa l'algoritmo di clock time adjustments.
- Il Clock adjust process esegue delle correzioni periodiche del (VFO) .

Network Time Protocol (NTP) (5)

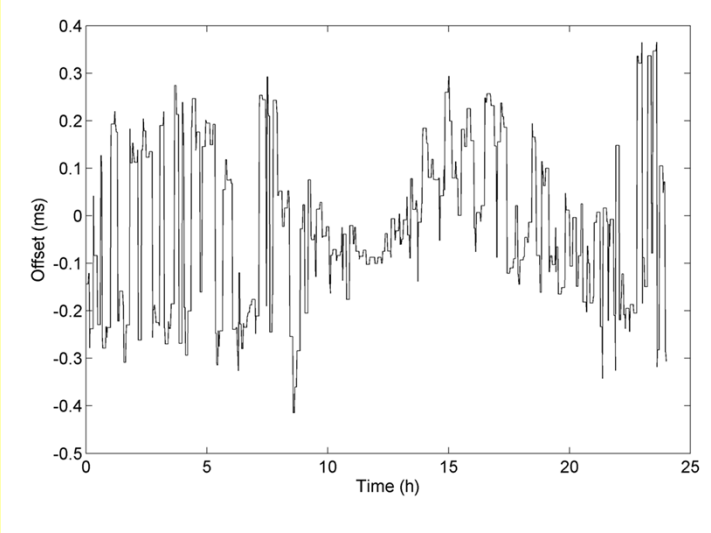
- Scambio di messaggi tra due processi NTP



Sono mostrati i timestamp numerati e scambiati tra i server A e B.
Se T_1,T_2,T_3,T_4 sono i valori dei quattro timestamp più recenti ,
se si assume $T_3 > T_2$,
se si assumono i clock di A e B stabili e con la stessa velocità,
se indichiamo $a = T_2 - T_1$ e $b = T_3 - T_4$:
se la differenza di ritardo di rete da A a B e da B ad A ,
detta "ritardo differenziale", è piccola, l'offset di clock θ e l'errore di round trip delay α di B relativo ad A al tempo T_4 sono tali che:

$$\theta = (a + b) / 2 \qquad \alpha = a - b$$

NTP (6): precisione



Il futuro di NTP: lo spazio

- La NASA e JPL stanno estendendo Internet per l'esplorazione spaziale creando IPIN: Interplanetary Internet.
- A causa dei lunghi tempi di propagazione, I protocolli internet esistenti quali TCP ed NTP non sono adatti.
- Comunque NTP sarà esteso con un algoritmo iterativo che modelli la meccanica celeste.

- Il programma di ricerca è stato finanziato da DARPA e NASA.
- L'obiettivo primario è l'esplorazione di Marte ed il supporto alla missione.
- Base stations di superfice e rovers dovranno eseguire esperimenti e collezionare dati.
- Satelliti in orbita marziana ritrasmettono dati e comandi fra le stazioni di superfice e la Terra

IPIN: Communication issues

- Le latenze di trasmissione fra la terra e Marte sono variabili ed in genere più lunghe rispetto a quelli delle internet sulla Terra e su Marte.
- Le velocità di trasmissione sono asimmetriche ed altamente variabili ed in generale molto più basse di quelle della Internet terrestre.
- La connettività fra la superfice di Marte e gli orbiters e fra la Terra e Marte non è continua, anche se è possibile prevedere i periodi favorevoli.
- Error recovery basato sulla ritrasmissione dei pacchetti non va bene. TCP è adatto alla Internet terrestre o Marziana, ma non per i collegamenti fra Terra e Marte.
- La dipendenza da database terrestri non è pratica su Marte per cui occorrerà pensare a database su Marte o vicini a Marte.

Sincronizzazione nelle applicazioni di misura

- In un sistema industriale possono esistere diversi elementi di misura ed attuazione.
- La consistenza fra le misure eseguite dai Sensori può dipendere dal tempo e dallo spazio.
- I nodi tipicamente eseguono misure di grandezze fisiche ad intervalli regolari e li inviano a dei concentratori che elaborano le informazioni ricevute.
- Operazioni più complesse, quali ad esempio il calcolo della velocità di oggetti mobili, è eseguito ordinando nel tempo gli eventi generati da un set di nodi.
- L'accuratezza della sincronizzazione deve essere tanto più elevata quanto più spinta è la dinamica del processo osservato. Piccoli errori di sincronizzazione possono indurre errori di calcolo considerevole.

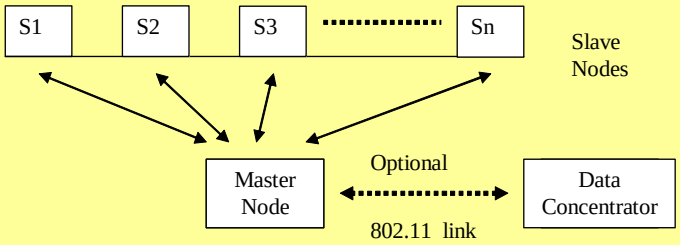
Un caso di studio come esempio

- Problema: localizzare la posizione di una sorgente sulla base del ritardo misurato dai sensori, usando uno specifico algoritmo di localizzazione.

- Esempio: in una caldaia, il rumore prodotto dalla fiamma indica che non sta lavorando correttamente. Non è però possibile capire in quale punto della caldaia il problema si presenta perchè la caldaia è sigillata.
- E' necessario localizzare il punto dove si generano le vibrazioni.

Architettura della Sensor Network

- Un gruppo di nodi wireless acquisiscono informazioni da sensori acustici o accelerometri e spediscono i segnali acquisiti ad un campionatore che spedisce la sequenza di informazioni ad un nodo Master.



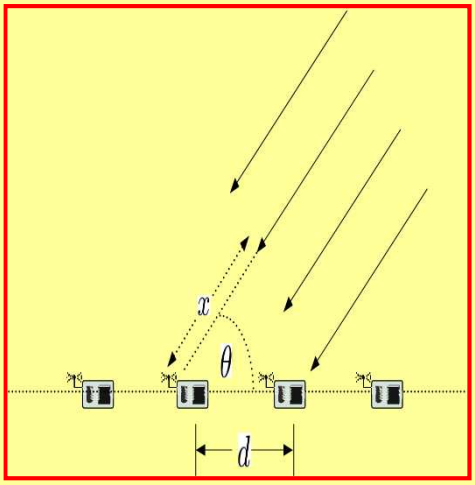
- Il Master processerà le informazioni ricevute e le spedirà (su un altro link wireless) ad un "Data concentrator" per ulteriori elaborazioni.

L'esempio

- Goal: stimare l'angolo di arrivo di un suono molto distante usando un array di sensori acustici.
- Dalla figura, θ può essere stimato quando x e d sono noti:

$$x = d \sin\theta$$

- d è noto a priori, x deve essere stimato dalle differenze nei tempi di arrivo
  - $x = C\,\Delta_t$ dove C è la velocità del suono.
  - Per d=1 m e Δ_t=0.001 otteniamo θ = 0.336 radians
  - Se Δ_t è stimato con un errore di 500 μs, la tima di θ può variare fra 0.166 e 0.518
- Morale: quello che sembra un piccolo errore nella sincronizzazione del clock può causare errori significativi nella stima degli angoli.



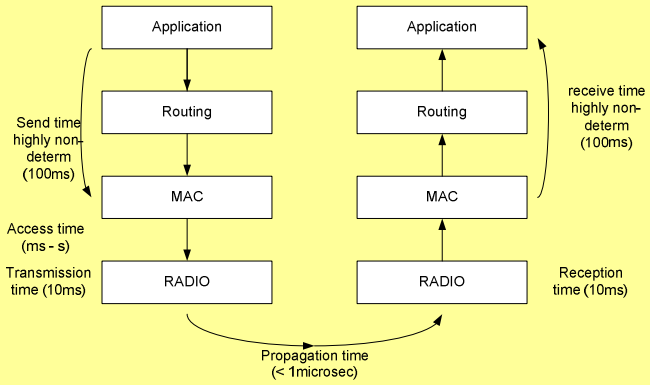
Problemi nelle Reti Wireless

Le reti wireless introducono ulteriori problemi nella sincronizzazone del clock rispetto alle reti wired:

- Difficoltà di propagazione
- Nodi nascosti
- Mobilità
- Disturbi elettromagnetici
- Consumi energetici
- Multi-hop

Fattori che influenzano la sincronizzazione

Latenza = Send_Time+ Access_Time
+ Transmission_Time + Propagation_Time
+ Reception_Time+ Receive_Time

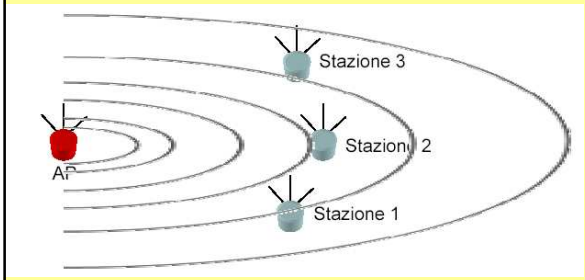


Fattori che influenzano la sincronizzazione

- *Send Time:* tempo richiesto per assemblare il messaggio e passare la richiesta al MAC layer.
- *Access Time:* ritardo dovuto all'attesa del canale.
- *Transmission Time: tempo impiegato dal trasmettitore radio per trasmettere un pacchetto bit a bit.*
- *Propagation Time*
- *Reception Time:* tempo impiegato dal ricevitore per ricevere l'intero messaggio.
- *Receive Time*: tempo impiegato per processare un messaggio in arrivo e notificarlo all'applicazione.

Sincronizzazione nativa in IEEE 802.11

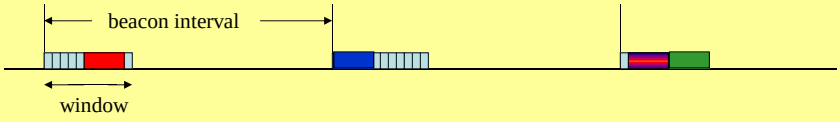
- In 802.11 è presente un meccanismo nativo di sincronizzazione usato dal protocollo MAC per coordinare le attività.
- L'AP periodicamente trasmette una frame chiamata "Beacon" frame.
- Tutte le stazioni periodicamente aggiornano il proprio clock sincronizzandolo con quello della stazione base (AP)



Il meccanismo aggiorna tutte le stazioni presenti nella cella coperta dall'AP

Sincronizzazione nativa in IEEE 802.11

- Il Tempo è diviso in beacon intervals, ciascuno contenente una beacon generation window.
- Ciascuna AP:
  - aspetta per un numero random di slots;
  - trasmette una beacon (se non lo ha già fatto qualche altra stazione).
- Beacon: è lungo alcuni slots.

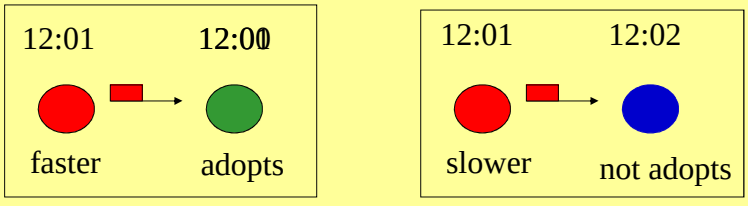


Sincronizzazione nativa in IEEE 802.11

- Ogni Beacon contiene un timestamp.
- Ogni stazione, appena ricevuto un beacon, ne utilizza il clock se

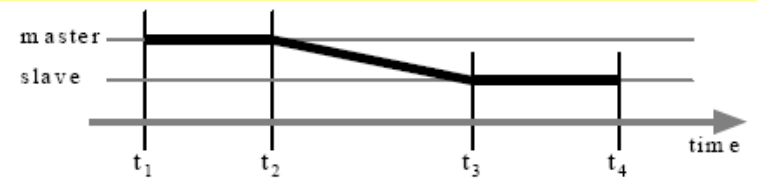
  T(beacon) > T(STA).

- I Clocks si muovono solo in avanti.



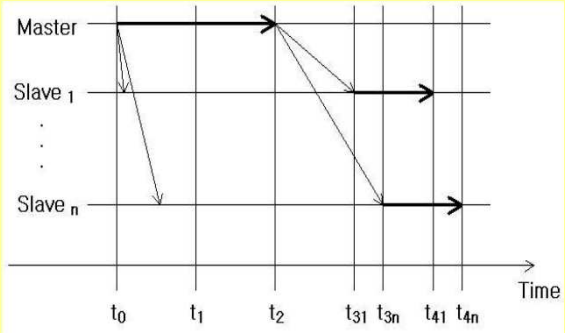
Sincronizzazione nativa in IEEE 802.11

- Il beacon contiene un time-stamp del clock locale del master che prende (t1) e trasmette (t2).
- Tutti gli slave ricevono fisicamente questa comunicazione al tempo t3 ed aggiustano i loro clock locali (t4).
- La precisione realizzata è limitata dalla variazione del *tempo critico di percorso* (tempo che intercorre tra il momento in cui il master prende il suo *time-stamp* e gli slave che aggiustano i loro clock locali, da t1 a t4).



Sincronizzazione nativa in IEEE 802.11

Due qualsiasi slave ricevono lo stesso frame approssimativamente allo stesso momento.

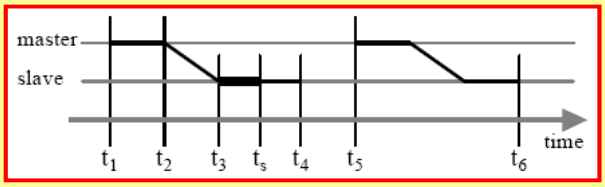


$|t31-t3n| \approx 0$

La precisione che può essere raggiunta attraverso questo protocollo è limitata dalla variazione del time-critical-path, cioè il tempo che intercorre dal momento in cui il master legge il proprio clock a quello in cui lo slave aggiusta il proprio, cioè l'intervallo di tempo $[t_1 , t_4]$.

Miglioramenti possibili nella sincronizzazione (1)

- E' possibile migliorare la precisione riducendo il tempo critico di percorso con la seguente procedura (M. Mock et alii):



•Il master prepara un messaggio di indicazione (t1) e lo trasmette (t2).
•Ogni slave riceve il messaggio di indicazione (t3) e memorizza il *time-stamp* locale (tS).
•Il master spedisce successivamente il suo *time-stamp* dell'ultimo messaggio di indicazione (t5).
•Ogni slave compara il *time-stamp* del master col proprio *time-stamp* dell'ultimo messaggio di indicazione ricevuto, calcola la differenza e aggiusta il suo clock locale (t6).

Miglioramenti possibili nella sincronizzazione (2)

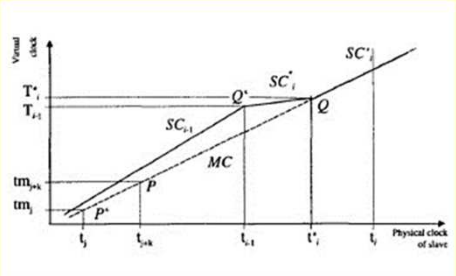
- Questo approccio ha comunque alcuni inconvenienti riguardo alle applicazioni real-time distribuite:
- una correzione diretta ed istantanea causa gaps nel tempo locale, portando ad una errata valutazione della misura dell'intervallo locale di sincronizzazione.
- una correzione istantanea può inoltre restituire valori negativi per gli intervalli di misura.
- Procedendo in questo modo si ha uno spreco di banda, poiché è necessario un numero di due messaggi per ogni ri-sincronizzazione.

Miglioramenti possibili nella sincronizzazione (3)

- Per ridurre il numero di messaggi trasmessi si inserisce il messaggio di verifica nel successivo messaggio utilizzato per la sincronizzazione.
- Il messaggio sm_{i+1} è utilizzato pure come messaggio di verifica di sm_i (i-esimo messaggio di sincronizzazione mandato dal master). In questo modo è sufficiente mandare un solo messaggio per ogni periodo di sincronizzazione.
- Robustezza rispetto al numero di messaggi persi: ciascun messaggio di sincronizzazione include, oltre al suo, i precedenti **n** valori di time-stamp.
- In questo modo, appena ricevuto un messaggio di sincronizzazione, ciascun slave può sincronizzare il proprio clock con quello del master se ha ricevuto almeno uno degli **n** precedenti messaggi di sincronizzazione, così da poter sopportare la perdita di **n-1** messaggi consecutivi.

Miglioramenti possibili nella sincronizzazione (4)

- Una sincronizzazione tempo-continua evita salti del clock"spalmando" la correzione sul prossimo periodo sincronizzazione .
- Occorre calcolare dei parametri che permettano l'aggiustamento dello SClock (clock virtuale, che dipende da quello fisico dello slave) sul MClock.

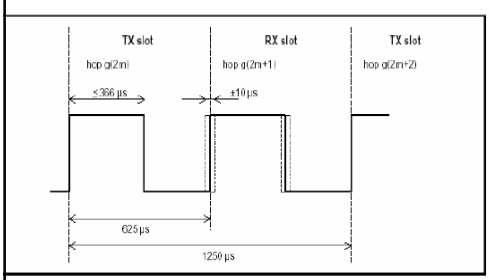


Occorre modificare la pendenza dello Slave clock per renderla uguale a quella del Master clock sfruttando i valori contenuti in messaggi successivi.

Sincronizzazione in Bluetooth

- Un meccanismo nativo di sincronizzazione è presente anche in Bluetooth.
- Il protocollo opera con un approccio Time slotted e richiede un perfetto allineamento degli slot temporali fra Master e Slave.
- L'allineamento del clock è eseguito durante la fase di page.
- La sincronizzazione ottenuta non può essere direttamente sfruttata dalle applicazioni poichè:
- - l'applicazione non ha accesso diretto al clock fisico (alcuni chip forniscono però un comando AT per leggere il registro contenente il clock).
- - il periodo è troppo lungo (312,5 microsec) e fornisce una granularità inadatta a molte applicazioni.

Sincronizzazione in Bluetooth



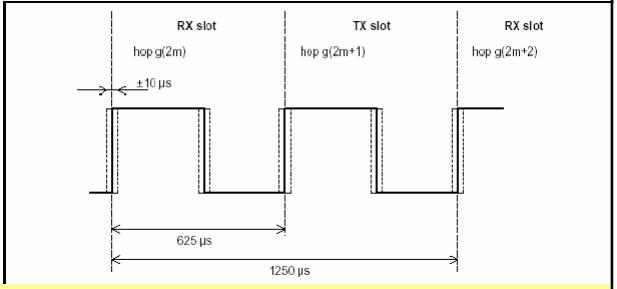
Il Master:

Non corregge mail il suo clock

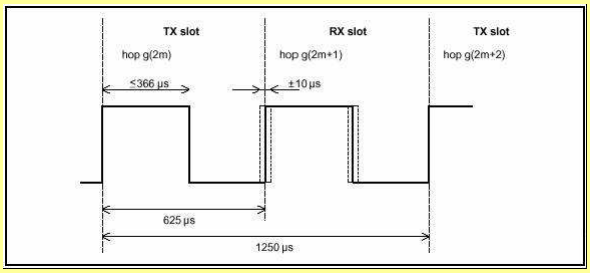
Tollera gli errori degli Slave

Lo Slave:

Corregge ad ogni pacchetto il suo clock stimato



La Sincronizzazione in Bluetooth



•Esiste un clock nativo e tutti gli slave sono sincronizzati col Master (errore inferiore al microsecondo).

Problemi:?

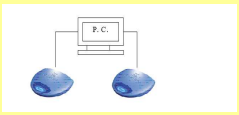
•La presenza dell'interfaccia HCI.

•Limitata risoluzione del clock (28 bits)

Sincronizzazione basata su un clock virtuale

- *Poichè la risoluzione del clock nativo è bassa, si può usare un clock virtuale realizzato come un processo applicativo* clock:
- Per sincronizzare il clock virtuale si può usare l'algoritmo di Cristian: un riferemento temporale comune può essere generato misurando il tempo Send/return di un messaggio.

- Uno Slave manda al Master la richiesta di sincronizzazione all'istante T1;
- Il Master riceve la richiesta e spedisce il suo clock Tm allo Slave;
- Lo Slave riceve il clock dal Master all'istante T2 e setta il suo clock al nuovo valore.



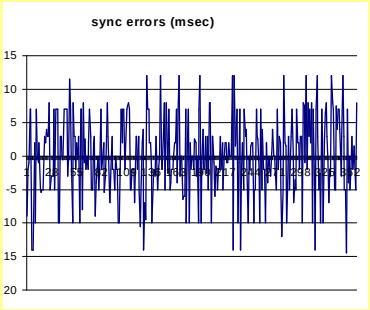
$$T_{new} = T_m + \frac{T_2 - T_1}{2}$$

Sincronizzazione dei clock

80

Errori di sincronizzazione

Gli errori di sincronizzazione dipendono dalla asimmetria fra i due tempi di trasmissione.



L'asimmetria deriva dai tempi di processamento non deterministici, e dai tempi di comunicazione in Bluetoot e nel PC.

Non è possibile sincronizzare il clock virtuale con quello nativo: ne consegue che i tempi di attesa per la trasmissione del clock sono impredicibili.

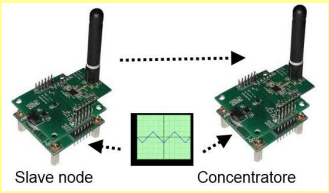
Si può migliorare l'algoritmo di Cristian?

- La precisione può essere migliorata effettuando la sincronizzazione con una media su diversi campioni.
- Lo Slave manda al Master la richiesta di clock synchronization all'istante T1;
- Il Master riceve la richiesta e spedisce il suo clock Tm allo Slave;
- Lo Slave riceve il Master clock all'istante eT2 e calcola il valore (T2 – T1)/2 del tempo di viaggio dei dati
- Calcola il valore medio
- Setta il nuovo valore

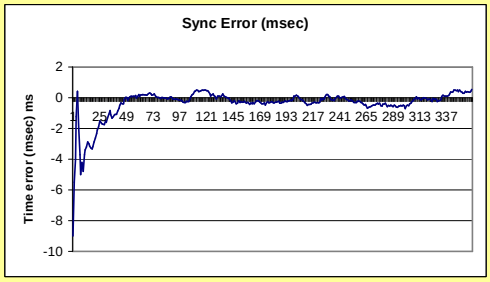
$$V_m = \frac{\sum_{i=1}^{N} \frac{T_{2_i} - T_{1_i}}{2}}{N}$$

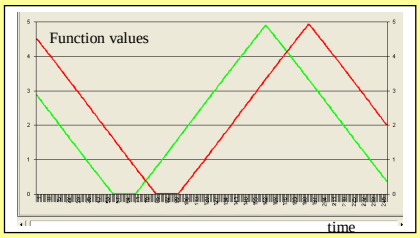
$$T_{new} = T_m + V_m$$

Si può migliorare l'algoritmo di Cristian?



- Appena acquisiti un certo numero di campioni, la precisione migliora
- L'errore è evidenziato dal confronto fra il segnale originato e quello ricostruito dal concentratore.





Clock Broadcast approach

- La principale causa di errore nel virtual clock è sopratutto il non-determinismo del *Send Time*, *Access Time* and *Receive Time* di Bluetooth.
- Una soluzione possibile: Reference Broadcast Synchronization (RBS) algorithm (Elstrin et alii): I nodi periodicamente spediscono un messaggio ai loro vicini usando una trasmissione broadcast al Physical level.
- I nodi riceventi utilizzano l'istante nel quale il messaggio arriva per registrare il loro tempo locale e quindi confrontare i propri clock locali.

Problemi della Reference Broadcast Synchronization

- Sono richieste operazioni di interpolazione basate sul metodo dei minimi quadrati e su operazioni di regressione, da eseguire sui dati di sincronizzazione.
- Lo scambio dei numerosi messaggi di sincronizzazione fra i vari Slave riduce la banda disponibile per la trasmissione di dati al Master
- Bluetooth è stato progettato per Master/Slave communication e non è adatto alla comunicazione Slave/Slave.

Master Reference Broadcast Synchronization

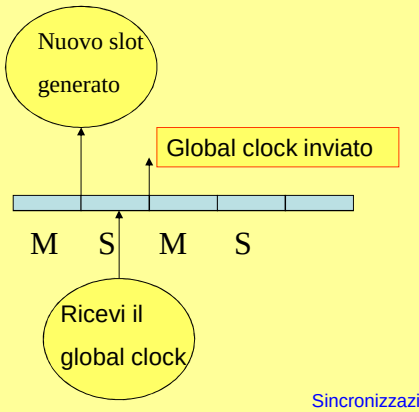
- Occorre modificare l'algoritmo RBS
- Tutti gli Slaves sono sincronizzati col Master virtual Clock mediante un messaggio Broadcast;
- Un time stamp (global clock) è contenuto nel messaggio broadcast;
- Appena un messaggio broadcast viene ricevuto dagli Slave, il global clock viene letto ed usato come Local_clock.

Ma

- *Send Time*, *Access Time* and *Receive Time* di BT sono ancora un problema.

Correzione del Send Time ed Access Time

- La sincronizzazione fra i processi applicativi (clock virtuale) e time slot, richiede uso di dispositivi BT dotati di porte GPIO per interagire velocemente con le applicazioni.



Quando è generato un nuovo slot il GPIO del Master produce un interrupt.

Nel Master, il Global_clock viene letto istantaneamente e passato al Baseband.

Il dato contenente il time_stamp (Global_clock) è trasmesso nel prossimo slot.

Viene così introdotto un offset costante di 625 microSec.

Correzione del Receive time

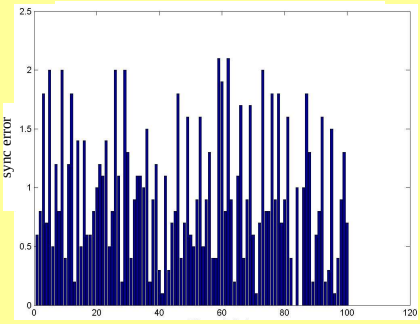
- Appena un messaggio di broadcast viene ricevuto da uno Slave, viene generato un interrupt per leggere tempestivamente il local_clock1.
- La lettura del messaggio ricevuto richiederà del tempo (receive time) che però non introdurrà errori poiché local_clock1 è stato letto istantaneamente.
- Appena il global clock è letto, lo Slave può calcolare il valore

(global_clock – local_clock1 +625)

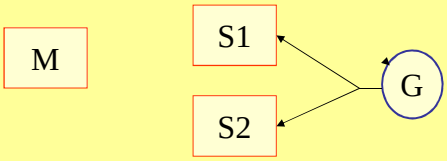
che viene sommato al valore corrente del local clock.

Performance evaluation

Layout of the STLC2450 Mini-Kit.



L'errore medio è inferiore al microsecondo



Problemi aperti

- Ancora una volta, una correzione diretta ed istantanea causa gaps nel tempo locale, portando ad una errata valutazione degli istanti di acquisizione delle variabili: continuous clock synchronization.
- Per una efficace sincronizzazione gli slave devono essere sempre attivi: elevati consumi energetici.

Altri algoritmi per le sensor networks

- In genere le WSN presentano maggiori problemi di sincronizzazione rispetto alle normali reti wireless poichè:
- La loro topologia è normalmente impredicibile
- La sorgente del clock ed i vari client possono distare diversi hop.
- Non sono previste periodiche manutenzioni Hw o Sw.
- La sincronizzazione introduce Communication Overheads
- I Servers devono essere sempre attivi e ciò contraddice la necessità di risparmiare le batterie.
- Le risorse di calcolo dei nodi sono limitate.

⟹ Sono necessari algoritmi specifici

Time-Sync Protocol per le WSN (TPSN)

- E' basato sul classico metodo di handshake di tipo Sender-Receiver
- I nodi sono strutturati in modo gerarchico come nel protocollo NTP
- I nodi si organizzano autonomamente in modo da comportarsi come server per alcuni nodi e come client nei confronti di un altro server
- I nodi di Livello 0 sono chiamati *root* node.
- La sorgente del tempo può essere esterna oppure uno dei nodi della sensor network.

Time-Sync Protocol per le SN (TPSN)

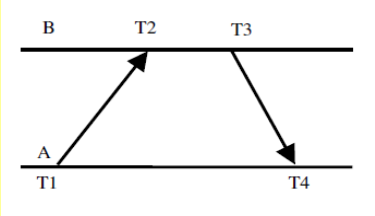
- Il root node non è fisso ma viene selezionato periodicamente usando un leader election algorithm
- Tutti I nodi possiedono un ID e ciascun nodo conosce i suoi vicini (Assunzione 1)
- Tutti i nodi hanno dei link bidirezionali coi propri vicini (Assunzione 2)
- Due fasi: Level Discovery e Sincronizzazione.

TPSN: Level Discovery Phase

- Un nodo del Root level invia in broadcast un "level discovery" packet ai suoi vicini
- Questi vicini si localizzano come slaves di livello 1 ed inviano un'altro pacchetto ai propri vicini indicando il proprio livello. Questo processo continua finchè tutti i vicini hanno assegnato un livello.
- In reti complesse può accadere che qualche nodo non riceva alcun pacchetto o che afferisca alla rete quando la fase di discovery è ormai conclusa. In tal caso esso aspetterà lo scatto di un time out e poi manderà un messaggio di "level request" ai propri vicini.
- Se un root node muore, I nodi del livello 1 non riceveranno alcun ACK da esso ed attenderanno fino allo scadere di un time out. Dopo di che inizieranno nuovamente un leader election algorithm per eleggere un Root Node.

TPSN: Synchronization Phase

- Viene effettuato un Two way message exchange fra sender e receiver



Il Clock Drift Δ ed il Propagation delay d, possono essere calcolati con le seguenti equazioni:

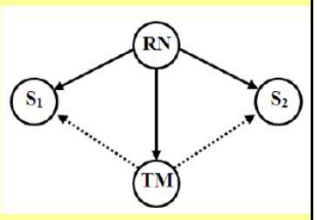
$$\Delta = \frac{(T2-T1)-(T4-T3)}{2}; d = \frac{(T2-T1)+(T4-T3)}{2}$$

TPSN: Synchronization Phase

- Il Root node inizia spedendo un time sync packet.
- Dopo un tempo random, i nodi di livello 1 si comporteranno da sender ed inizieranno un two-way message exchange con il root, per sincronizzarsi.
- Ogni nodo del livello 1 spedirà un ack dopo essersi sincronizzato col root node.
- I nodi di livello inferiore ascolteranno questo scambio di messaggi ma attenderanno un tempo random (per essere certi che tutti i nodi di livello superiore si siano sincronizzati) prima di iniziare a loro volta a sincronizzarsi coi nodi adiacenti.

Dynamic Continuous Clock Synchronization

- L'obiettivo è sincronizzare tutti gli Slave con il Master con un protocollo semplice che usa pochi messaggi di sincronizzazione e che può girare anche su dispositivi semplici permettendo lunghi sleeping intervals.
- Il Time Master (TM)fornisce il clock di riferimento.
- Elemento chiave è il Reference Node (RN) che periodicamente spedisce dei Reference Broadcast Messages RBM).

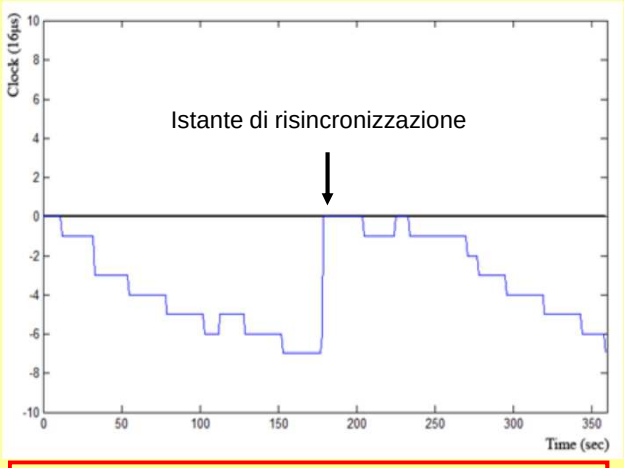


- Quando l'i-esimo reference message è ricevuto, (approssimativamente nello stesso istante $t_{RBM(i)}$), il TM e gli slaves, leggono ed immagazzinano I loro clock. Successivamente il TM broadcasts il valore del suo timestamp che sarà usato da tutti gli slaves per calcolare l'offset fra I loro clock e quello del nodo TM.

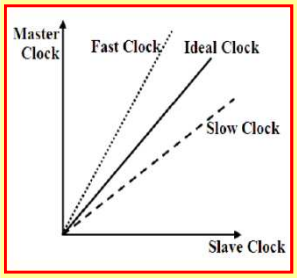
$$Offset_i(TM,S) = C_{TM}(t_{RBM(i)}) - C_S(t_{RBM(i)})$$

97

Accumulo dell'errore del clock nel DCCS



Differenza fra il clock del Master e di uno Slave (Sync. Interval 180 sec.,)



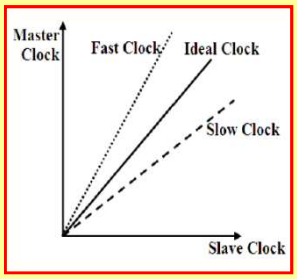
A causa del drift fra i clock, gli slave accumulano un errore crescente e devono essere risincronizzati periodicamente.

98

Dynamic Continuous Clock Synchronization

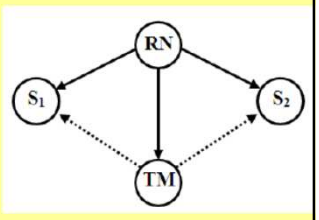
Per tenere in conto gli errori dovuti al drift, occorre considerare lo Skew fra gli slave ed il TM. Calcoliamo il coefficiente:

$$\alpha_i = \frac{C_{TM}(t_{RBM(i)}) - C_{TM}(t_{RBM(i-1)})}{C_S(t_{RBM(i)}) - C_S(t_{RBM(i-1)})}$$



Il coefficiente α aggiornato in maniera pesata ad ogni nuova sincronizzazione:

$$\overline{\alpha_i} = \frac{k * \alpha_{i-1} + \alpha_i}{k+1}$$



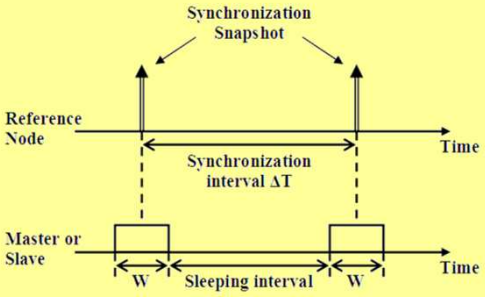
Possiamo quindi ricalcolare il virtual clock in ciascun slave, al generico istante t compreso fra due sincronizzazioni:

$$VC_S(t) = \overline{\alpha_i} * (C_S(t) - C_S(t_{RBM(i)})) + C_{TM}(t_{RBM(i)})$$

Problemi nel DCCS

- Il Reference node (RN) non è sincronizzato con gli altri nodi.
- Gli altri nodi non possono conoscere l'istante in cui ciascun Reference Broadcast Message sarà inviato.

W=2 ΔT *δ
δ clock drift
ΔT sync interval



- Inoltre, RN è un single point of failure

DCCS Improvements (1/3)

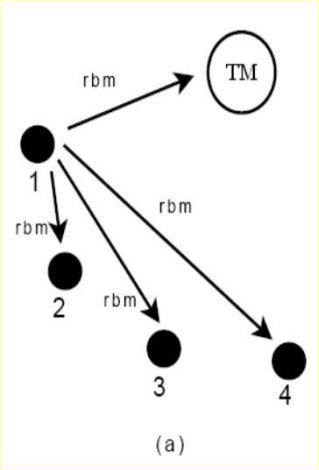
- Il protocollo può essere modificato in modo che a turno ciascun nodo svolga il ruolo di RN.
- In tal modo tutti i nodi si sincronizzeranno col Master, saltando un solo ciclo di sincronizzazione (quando svolgono il ruolo di RF).
- La robustezza del sistema viene fortemente aumentata.
- Con tale modifica anche il reference node viene sincronizzato.

Sono necessarie due fasi:

- Una fase di inizializzazione, durante cui tutti gli Salve si registrano col nodo Time Master.
- Una fase operativa, in cui successivi round di sincronizzazione si alternano con la normale attività della rete.

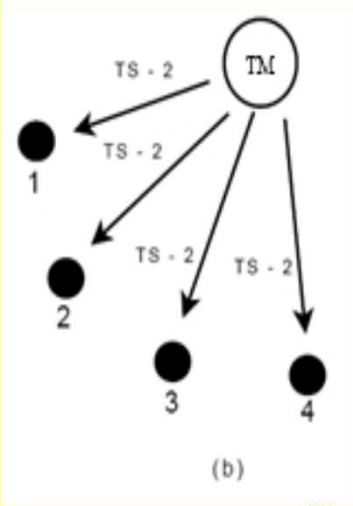
DCCS Improvements (2/3)

- Durante la fase operativa si svolgono le seguenti operazioni:
- 1. Il RN designato. spedisce in broadcast un messaggio RBM.
- 2. Sia il Time Master che gli slaves, alla ricezione del messaggio RBM effettuano un timestamp dei loro clock locali.



DCCS Improvements (3/3)

- 3. Il Time Master spedisce in broadcasts un messaggio col proprio timestamp, l'indirizzo del prossimo RN e l'istante in cui la prossima sincronizzazione dovrà essere eseguita.
- 4. Gli Slave usano il timestamp del Master per sincronizzare i propri clock.
- 5. Il RN designato, schedula la trasmissione del prossimo RBM.



Implementazione sudisposivi reali

• Xbee-Pro transceiver

• semplice microcontrollore a 8 bit MC9S08GT60

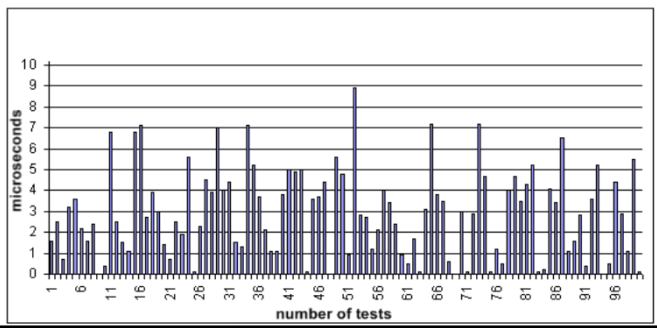
•61KB RAM, 4KB Flash memory

•MaxStream Starter Development Kit

•Freescale BeeKit Wireless Connectivity Toolkit

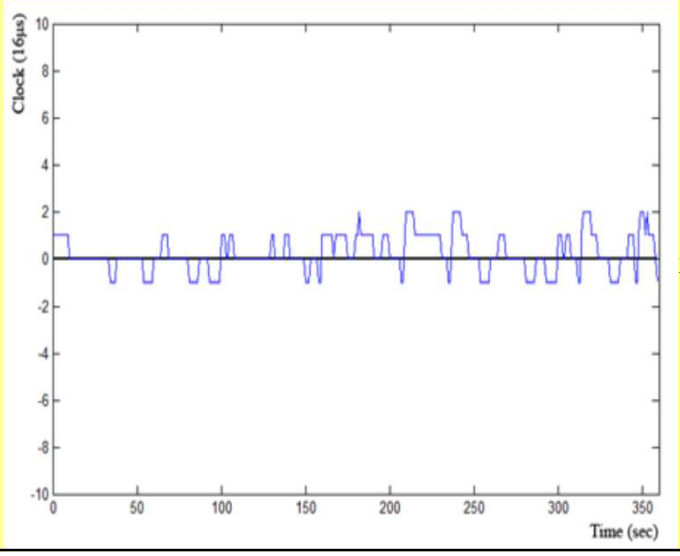
Performance evaluations (1)

- Una misura essenziale per valutare la precisione ottenibile è la differenza fra i ritardi con cui l'interrupt generato dalla ricezione di un RM è servito dal syncronization middleware in ogni nodo.
- Ciò permette di valutare l'effetto, sul critical path, dei ritardi di computazione in ogni nodo.



105

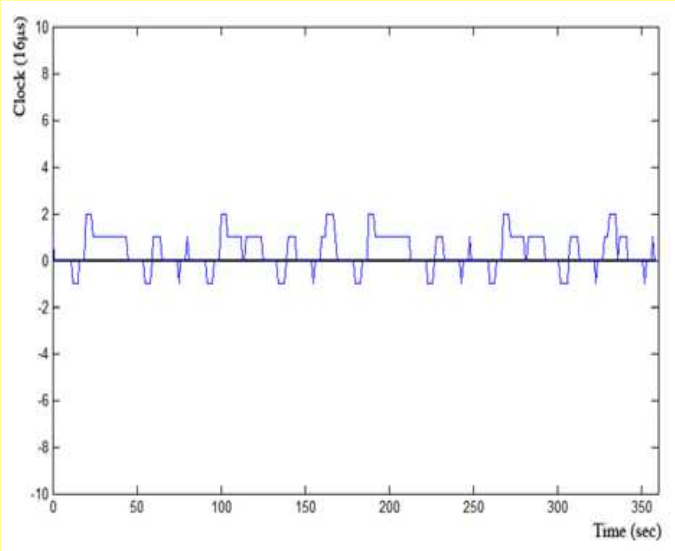
Performance evaluations (2)



Synchronization interval : 20 s

106

Performance evaluations (3)



Synchronization interval : 60 s

107

Considerazioni sul DCCS

- Il protocollo DCCS fornisce una sincronizzazione di clock con un errore di pochi microsecondi.
- La precisione ottenibile è influenzata dalla durata del tick fisico nei nodi e dalla stabilità a breve termine dei clock locali.
- Può tollerare il crash di uno o più slave.

open issues:

- Occorre definire la strategia da adottare in caso di perdita di sincronizzazione dovuta ai disturbi.
- Non è stato valutato il comportamento del protocollo in contesti multi-hop.

108

Sincronizzazione dei clock