UNIVERSITÁ DEGLI STUDI DI MODENA E REGGIO EMILIA

Facoltà d'Ingegneria

Corso di Laurea in Ingegneria Informatica

TINYDB: GESTIONE DEI DATI IN RETI DI SENSORI

Relatore:
Chiar.mo Prof. Sonia Bergamaschi

Candidato:
Giorgia Loschi

Anno Accademico 2006/2007

Parole chiave:

TinyDB

Sensor Networks

INDICE

INTRODUZIONE

Negli ultimi anni il progresso compiuto nel campo della miniaturizzazione degli apparati di elaborazione, memorizzazione e comunicazione ha permesso la creazione di nuove famiglie di dispositivi che, per le loro caratteristiche, possono essere distribuiti in grandi quantità nell'ambiente per svolgere funzioni di rilevazione dati, di controllo della movimentazione delle merci e in generale per realizzare nuove e avanzate forme di interazione con il mondo circostante. Esempi d'uso si trovano nella logistica, nei sistemi di telecontrollo ed in generale nelle applicazioni di "ambient intelligence". Tra gli esempi più noti e interessanti di questi sistemi periferici intelligenti e pervasivi troviamo le reti wireless di sensori (*Wireless Sensor Network*, WSN). Su di esse si concentra lo studio riportato in questo elaborato. Sarà fornita una panoramica sugli aspetti più salienti di questa nuova tecnologia, concentrandosi poi, nel secondo capitolo, sullo studio di TinyDB, un sistema per la gestione dei flussi di dati nelle reti di sensori. TinyDB fornisce un'interfaccia SQL-like che consente all'utente di interrogare la rete di sensori come se fosse un database tradizionale. Vedremo quindi gli algoritmi adottati dal sistema per l'ottimizzazione, la disseminazione e l'esecuzione delle interrogazioni.

Capitolo 1

LE RETI DI SENSORI

1.1 Introduzione alle reti di sensori

Il mercato dei sensori è estremamente variegato, grazie all'elevato numero di dispositivi prodotti e al loro utilizzo in ogni ambito dell'attività industriale.
Le aziende produttrici sono alla continua ricerca di nuove tecnologie per realizzare dispositivi caratterizzati da costi contenuti che siano, al tempo stesso, precisi, affidabili e rispondano alla continua e crescente domanda di nuove funzionalità per applicazioni sempre più sofisticate. Una tipica rete di sensori può essere costituita da un numero elevato di nodi collegati fra loro mediante cavi multipli. Ogni nodo è dotato di un microprocessore e di uno o più sensori collegati mediante interfacce proprietarie.
In particolare lo standard IEEE 1451 *Standard for Smart Transducer Interface for Sensor and Actuators*, cerca di stabilire una serie d'interfacce comuni per connettere fra loro sensori con dispositivi a microprocessore, oltre a definire il prototipo di uno *smart sensor* indipendentemente dalla rete all'interno del quale questo sarà inserito.
In primo luogo è necessario dare una definizione esauriente di smart sensor. In letteratura se ne trovano diverse, ma probabilmente la più esauriente è quella proposta dallo standard IEEE 1451.2:
"Un trasduttore che integra le funzioni necessarie alla corretta rappresentazione della grandezza misurata o controllata. Tipicamente queste funzionalità sono in grado di semplificare l'integrazione del trasduttore in applicazioni che utilizzino strutture di rete".
Lo standard utilizza il termine *transducer* per indicare contemporaneamente sensori ed attuatori; all'interno dell'elaborato si utilizzeranno i termini smart sensor e sensore.

1.2 Cenni storici

Le *Wireless Sensor Networks* (WSN) furono utilizzate per la prima volta dalla difesa americana durante la Guerra Fredda per monitorare gli spostamenti dei sottomarini sovietici *SOund SUirveillance System* (SOSUS) e sono tutt'ora utilizzati per monitorare i fondali oceanici.

Le moderne ricerche condotte su reti di sensori senza fili iniziarono nel 1980 con le DNS (Reti di sensori distribuiti) progettate da DARPA (*Defence Advanced Research Projects Agency*). Da questo progetto nasce il primo sistema operativo *communicate-oriented*, il quale supportava collegamenti trasparenti, riconfigurazione del sistema e il *rebinding*.

Un primo progetto importate fu portato avanti dal MIT (*Massachusetts Institute of Technology*), questo progetto consisteva in particolari elaborazioni di segnali acustici per il rilevamento di elicotteri e aerei, basandosi sul ritardo di propagazione dei segnali acustici.

Uno dei più grandi ostacoli a quel tempo erano le dimensioni dei sensori, proprio perché lo sviluppo di questa tecnologia presuppone una certa maturità di svariate discipline.

I recenti progressi tecnologici nei sistemi *micro-elettro-meccanici* (MEMS) nelle comunicazioni wireless e nell'elettronica digitale hanno permesso la realizzazione di nodi piccoli a bassa potenza e a costi contenuti.

1.3 Reti di sensori wireless

Obiettivo di questo paragrafo è quello di dare una visione d'insieme delle reti di sensori, ponendo a confronto le tue tipologie esistenti: reti *wired* e reti wireless.

La trattazione si concentrerà quindi sulle caratteristiche delle reti senza fili, discutendone gli aspetti di maggior interesse, ma anche i vincoli che fanno di queste reti uno strumento molto potente per l'acquisizione e la gestione dei dati, ma che, al tempo stesso, richiede importanti valutazioni in fase di progetto.

1.3.1 Reti cablate: vantaggi e limiti

Convenzionalmente, le comunicazioni fra nodi sensore ed i controllori centralizzati, così come le prime applicazioni degli smart sensor, prevedono interfacce di comunicazione cablate. L'utilizzo dei cavi consente l'impiego di dispositivi che non hanno limitazioni di potenza poiché, se è possibile una connessione cablata per i dati, allora sarà in generale possibile prevedere anche una o più linee di alimentazione; queste soluzioni consentono inoltre buoni livelli di sicurezza dal momento che bisogna avere accesso fisico diretto al cavo per poter prelevare informazioni dalla rete.
Al contempo presentano però gravi limitazioni:

- Impossibilità o difficoltà d'installazione in ambienti inospitali per l'uomo

- Alti costi d'installazione: l'installazione di ciascun nodo richiede manodopera e materiali per le operazioni di cablaggio.

- Struttura rigida: risulta difficile aggiungere nuovi nodi alla rete o modificare la posizione di sensori preesistenti senza riconsiderarne l'intera struttura.

1.3.2 Requisiti delle reti wireless

Le soluzioni *Wireless* sembrano essere la soluzione ideale ai problemi delle reti cablate, ma comportano al tempo stesso una serie di svantaggi in termini di:

- problemi di propagazione del segnale
- interferenze
- sicurezza
- requisiti di potenza
- norme legislative

Per molti di questi problemi esistono soluzioni efficaci, ma per ognuna è necessario tener conto dell'aumento della complessità progettuale e del conseguente aumento dei costi di realizzazione. Molte applicazioni, infatti, non consentono di utilizzare soluzioni *wireless* evolute come ad esempio i sistemi di telefonia cellulare o le reti descritte dagli standard IEEE 802.11 *Wireless Local Area network*, o altre soluzioni ancor più costose.

Delineando quali siano i **requisiti** a cui una rete di sensori wireless deve rispondere, si osserva che alcuni risultano essere comuni a qualsiasi tipologia di rete *wireless* (prestazioni, *range*, sicurezza e consumi); altri invece dipendono dalla particolare applicazione: è il caso, ad esempio, della determinazione della velocità di trasmissione, dal momento che alcune applicazioni richiedono decine di megabit al secondo (es. Applicazioni di video sorveglianza), mentre altre hanno requisiti meno stringenti nell'ordine di pochi kBit al secondo (es. Telecomandi, sensori di temperatura ecc.).

Le **dimensioni** delle reti stesse possono variare in funzione dell'applicazione, da un metro fino ad alcuni chilometri. Infine, è opportuno analizzare le caratteristiche peculiari che differenziano le reti di sensori dalle reti tradizionali, come la necessità di prevedere modalità di autoconfigurazione.

Scopo principale di una rete di sensori è distribuire sulla rete le informazioni raccolte da ciascun nodo. Affinché i dati possano essere verificati e coordinati, l'utente deve essere in grado di accedere ad ogni dispositivo per conoscerne la calibrazione effettuata dal costruttore ed ogni dato utile alla sua identificazione. Risulta infine indispensabile che siano previste delle modalità di identificazione di ciascun nodo all'interno della rete.

Altro **fattore caratterizzante** le reti di sensori è la necessità di raccogliere le informazioni dai sensori in modo sincrono. Si ha cioè l'esigenza di sapere esattamente quando una grandezza viene rilevata da un sensore, e, in generale una rete di sensori deve avere una organizzazione più deterministica dal punto di vista dei tempi, rispetto a reti ad accesso casuale, generalmente impiegate nelle reti informatiche.
Ogni livello fisico in grado di realizzare uno standard simile o aderente all' IEEE 1451 deve prevedere un robusto meccanismo di sincronizzazione, per permettere la

coesistenza dei diversi dispositivi sulla rete visto che la risoluzione temporale nelle applicazioni più stringenti può essere nell'ordine del microsecondo.
Dalle osservazioni raccolte fino a questo punto si desume che la realizzazione di una rete di sensori *wireless* richiede l'utilizzo di tecniche di rete specifiche. Anche se molti protocolli e algoritmi sono stati proposti in letteratura per realizzare reti *wireless ad hoc*, questi non sono completamente adattabili alle reti di sensori *wireless*, a causa delle specifiche necessità di questo tipo di rete.

1.3.3 Reti wireless vs. Reti ad hoc

Si intende per rete *ad hoc* una infrastruttura di rete che non richiede un coordinatore centrale ed in cui ogni nodo riveste il duplice ruolo di nodo e *router*.
Riassumiamo le principali caratteristiche dei nodi all'interno di una WSN*:*

- Il loro numero può essere di alcuni ordini di grandezza superiore rispetto a quelli presenti in una rete ad hoc;

- Sono posizionati con densità spaziali molto elevate (decine o centinaia di sensori nello spazio di pochi metri);

- Possono incorrere malfunzionamenti che non devono pregiudicare l'efficienza della rete;

- La topologia di rete può variare nel tempo in modo molto frequente;

- Utilizzano principalmente comunicazioni di tipo *broadcast*[2];

- Presentano stringenti limiti in termini di potenza;

A causa dell'elevata densità di posizionamento, i nodi possono essere molto vicini tra loro e questo comporta un pregio dal momento che si potrebbero realizzare algoritmi di

rete *multihop*[1] per raggiungere il corretto destinatario dell'informazione, ma, al tempo stesso, potrebbero insorgere problemi di mutua interferenza fra sensori distinti.
L'uso di strategie *multihop*[1] può consentire l'utilizzo di basse potenze di trasmissione, migliorando le caratteristiche dei nodi in termini di requisiti di potenza. Quest'ultima risulta infatti essere uno dei vincoli maggiori poiché i nodi utilizzano delle sorgenti di potenza (spesso batterie AA) che non possono essere in generale sostituite, o, in ogni caso, non possono essere sostituite frequentemente. Per questo motivo una efficiente implementazione del *wireless sensor networking* deve prevedere meccanismi che diano la possibilità all'utente di scegliere il compromesso migliore fra prestazioni e durata delle batterie.

1.3.4 Estensione delle reti wireless

Lo sviluppo di reti wireless dedicate al mondo dei sensori ha visto nell'ultimo periodo l'affermazione delle *Low rate Personal Area Network* (LR-WPAN). Queste reti, come si può intuire, sono caratterizzate da dimensioni contenute e conseguentemente da bassi *transfer rate* (tassi di trasmissione); la definizione stessa di LR-WPAN risulta essere in netto contrasto con le metriche con cui generalmente siamo abituati a valutare una rete cioè QoS e *data rate*, caratteristiche che risultano essere aspetti secondari in una rete di sensori poiché, come già detto, molte applicazioni richiedono semplicemente il trasferimento di pochi kBit/s.
Fattori di primaria importanza diventano quindi il contenimento dei costi e dei consumi.
Nello schema riportato di seguito, le LR-WPAN sono inserite all'interno della generica classificazione delle reti in funzione dell'area coperta. Si osserva che, anche considerando aree ristrette, il numero di nodi può essere comunque molto elevato.

[1]Si ha comunicazione *multihop* quando un messaggio non viene trasmesso direttamente dal mittente al destinatario, ma per mezzo di un insieme di nodi, ognuno dei quali riceve un messaggio e lo ritrasmette al vicino, formando un percorso che congiunge il mittente con il destinatario. La trasmissione *multihop* permette di coprire lunghe distanze pur avendo a disposizione apparati di comunicazione a basso consumo energetico in grado di coprire solo piccole distanze.

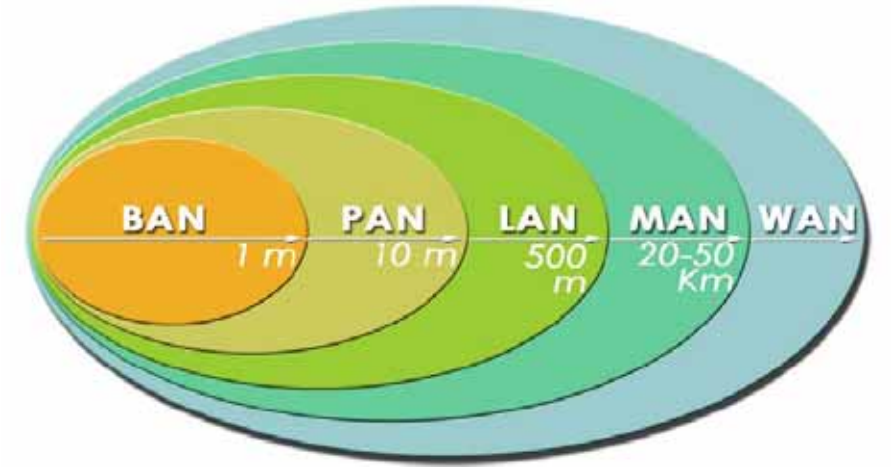


Figura 1: classificazione delle reti in funzione dell'area coperta

Nella Figura, vengono riportate le dimensioni caratteristiche delle reti

- WAN (*Wide Area Network*)
- MAN (*Metropolitan Area Network*)
- LAN (*Local Area Network*)
- PAN (*Personal Area Network*)
- BAN (*Body Area Network*)

Questo tipo di classificazione viene in genere ricondotto alla definizione di reti cablate, anche se, in generale, è possibile costruire un modello molto simile in cui inserire le diverse tipologie di rete wireless ad oggi presenti sul mercato.

1.3.5 Vantaggi rispetto alle reti cablate

L'interesse nei confronti delle reti di sensori wireless è motivato da tre aspetti principali:

1. Necessità di ridurre i costi d'installazione dei sensori, dovuti ai materiali, ai cavi impiegati, alla fase di *testing* e di verifica. Per esempio, un fine corsa può costare tranquillamente meno di un dollaro, ma i suoi costi d'installazione possono raggiungere i 50$ o anche più, a causa delle difficoltà di installazione.

2. I cavi necessitano dell'utilizzo di connettori che per loro natura sono soggetti ad usura, possono rompersi, o sconnettersi.

3. Il terzo aspetto è il fatto che le WSN permettono la realizzazione di ambienti, con un'alta densità di sensori, che implementano una struttura di controllo che genera grandi quantità di informazioni.

Le corrispondenti versioni cablate, anche nei casi in cui fosse fisicamente possibile realizzare una rete dotata di un elevato numero nodi, risultano essere svantaggiose da un punto di vista economico e ciò rende la realizzazione di sistemi ad elevate densità di nodi-sensore cablati praticamente irrealizzabile, o comunque economicamente improponibile.

Una soluzione *wireless* consente maggior flessibilità, semplicità di installazione e manutenzione, ma al contempo bisogna tenere conto del fatto che i vantaggi appena descritti, in molte applicazioni, non sono sufficienti per prevedere la completa sostituzione delle reti cablate.

1.4 Scelte tecnologiche

In questo paragrafo si prendono in considerazione alcuni dei fattori che il progettista di reti wireless deve considerare in fase di pianificazione per effettuare una scelta tra le tecnologie disponibili.

1.4.1 Fault Tolerance

Alcuni nodi della rete possono smettere di funzionare a causa di guasti fisici, di interferenze o per l'esaurimento delle batterie. In questi casi la rete deve essere in grado di riconfigurarsi per escludere il nodo guasto e ripristinare i collegamenti.
Con *fault tolerance* (tolleranza al guasto) viene indicata la capacità di una rete di sensori di mantenere le sue funzionalità anche in presenza di malfunzionamenti nei suoi nodi.

La tolleranza ai guasti (o *affidabilità*) $R_k(t)$ nel nodo k segue una distribuzione di Poisson ed è la probabilità di non avere guasti a un certo tempo t.
Nell'intervallo (0,t) assume quindi la seguente espressione:

$$R_k(t) = e^{-\lambda kt}$$

Dove λ_k è il tasso di fallimento del nodo k.

In fase di progettazione di una WSN, si deve tenere presente quale sia la tolleranza richiesta e quindi rispettarla implementando opportuni algoritmi.

1.4.2 Scalabilità

È la proprietà della rete di potersi espandere, acquisendo nuovi nodi in momenti successivi. Questa proprietà è essenziale sia perché ogni nodo ha una vita limitata, sia perché è possibile aggiungere ulteriori nodi per sostituire quelli guasti.
La densità della rete quindi varia velocemente a seconda del tipo di applicazione e della condizioni in cui si trova, quindi ogni singolo sensore deve essere in grado di adattarsi alla particolare situazione.
La densità è il numero di nodi presenti all'interno del raggio di trasmissione di un nodo e può essere calcolata nel seguente modo:

$$\mu(R) = (N\pi R^2)/2$$

dove N è il numero di nodi distribuiti in una regione di area A ed R è il raggio di trasmissione di un nodo.

1.4.3 Costi produttivi

Come si è già sottolineato in precedenza, uno dei vantaggi dell'uso di reti di sensori wireless è la possibilità di utilizzare un numero molto elevato di mote. Affinché questo risulti essere effettivamente un vantaggio, si deve garantire che il costo di ciascun dispositivo sia molto contenuto; infatti se il costo del numero di nodi richiesti supera il costo di una normale struttura cablata, i vantaggi tecnologici risultano essere annullati dallo svantaggio economico.
La tecnologia allo stato dell'arte dovrebbe consentire di avere nodi sensore dal costo contenuto al disotto del dollaro; ad oggi però, se si considerano per esempio dispositivi realizzati utilizzando come sistema di comunicazione Bluetooth, il costo della sola radio supera l'obiettivo che è stato prefissato.
Il problema del costo inoltre non risiede solo nella tecnologia di comunicazione scelta, ma deriva anche dai trasduttori, dall'elettronica di condizionamento del segnale ed infine dalle unità di calcolo.
Il progetto di uno smart sensor dovrà quindi essere guidato dalle particolari richieste dell'applicazione, scegliendo i componenti che garantiscano le specifiche di progetto, ma che al contempo rispettino le disponibilità di budget.

1.4.4 Ambienti operativi

I sensori possono essere utilizzati in luoghi fortemente inospitali, quali per esempio i campi di battaglia, oppure all'interno di processi produttivi caratterizzati da alte pressioni, alte temperature, forti vibrazioni, ecc. Il progetto di uno smart sensor quindi non può prescindere dal contesto all'intero del quale dovrà essere utilizzato, in particolare è necessario prevedere dei *package* in grado di sopportare le condizioni di stress tipiche dell'ambiente in cui deve lavorare.
In fase di progettazione è necessario fare tali valutazioni senza tuttavia dimenticare il soddisfacimento dei requisiti di costo e di tolleranza al guasto.

1.4.5 Topologia della rete

Col termine *topologia di rete* si indica la posizione che i diversi dispositivi vengono ad occupare nello spazio.

Per quel che concerne il posizionamento fisico dei mote è opportuno ricordare che uno dei vantaggi delle reti wireless risiede proprio nell'estrema libertà con la quale si possono collocare i nodi sensore; si è già ricordato come la densità tipica di sensori possa essere nell'ordine delle decine per metro quadrato e come la posizione relativa fra i diversi dispositivi possa evolvere nel tempo. Ricordando inoltre che anche nodi sensore essenzialmente statici, cioè posti in posizioni precise che non evolvono nel tempo sono soggetti al problema dello spegnimento a causa della mancanza di energia, ciò comporta che anche questi dispositivi contribuiscono all'evoluzione della topologia di rete. In senso inverso la scalabilità stessa della rete contribuisce a farla evolvere da un punto di vista topologico.

Le osservazioni fatte portano a valutare l'utilizzo di *topologie funzionali di rete* e di protocolli di *routing* (protocolli di instradamento) che ne garantiscano l'affidabilità anche in corrispondenza di continui cambiamenti di posizione dei nodi e/o dell'aggiunta/rimozione dei nodi stessi.

In prima approssimazione è possibile classificare le topologie di rete in tre diversi gruppi: reti a stella, reti *mesh* o *peer to peer* ed infine reti ad albero.

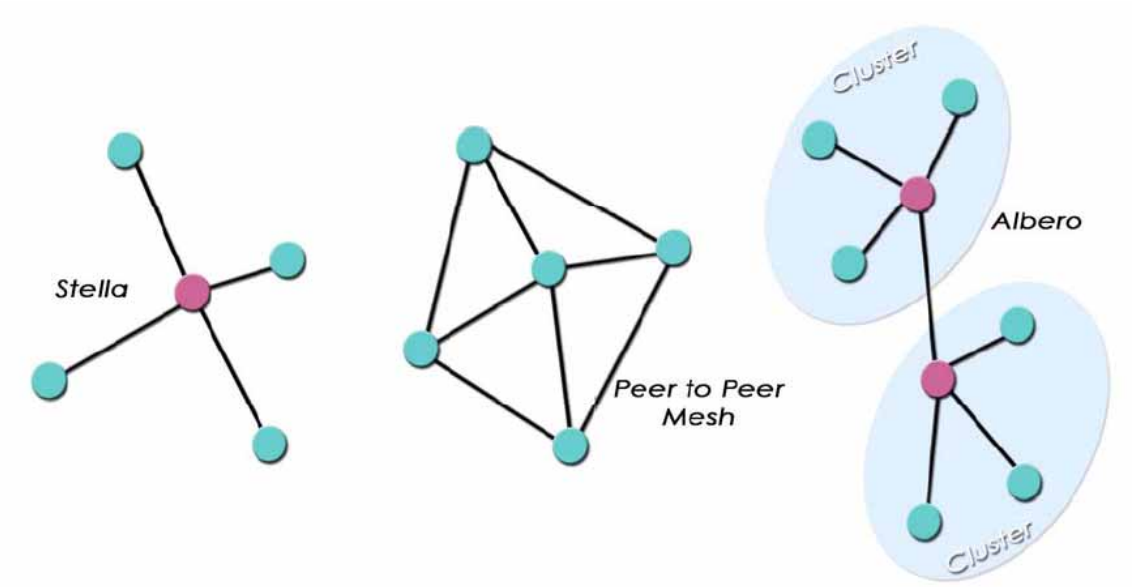


Figura 2 Alcune topologie di rete

- **Rete a stella** Presenta un nodo centrale (centro della rete) con funzionalità di coordinatore e verso il quale fanno riferimento tutti gli altri nodi. Ciò implica che affinché due nodi possano comunicare tra loro è necessario che entrambi comunichino con il coordinatore della rete.

  Vantaggi:

  - molto semplice da implementare
  - consente l'impiego di protocolli poco onerosi da un punto di vista computazionale: è valida nel caso in cui un nodo (il coordinatore) non abbia particolari vincoli in termini di potenza mentre tutti gli altri debbano essere alimentati da un'unità di potenza limitata.
  - il coordinatore può anche funzionare da *bridge* verso altri sistemi di connessione

- **Rete peer to peer** In questo tipo di rete non è essenziale il ruolo del coordinatore poiché ogni dispositivo è in grado di connettersi con tutti gli altri. Si rende quindi possibile la realizzazione di percorsi ridondanti.

  Vantaggi:

  - consentono di aumentare l'affidabilità della rete

  Svantaggi:

  - richiedono l'implementazione di algoritmi di routing più complessi.

- **Topologia ad albero** Diversi *cluster* (grappoli) costituiti da gruppi di nodi possono interconnettersi in modo simile a come avviene per la diramazione delle foglie su un albero. Ciascun cluster, infatti, è dotato di un nodo principale che rappresenta il punto d'accesso per la sottorete in questione.

  Vantaggi:

  - riduzione dei percorsi di comunicazione possibili
  - sviluppo di sistemi di gestione meno complessi.

La costruzione e il mantenimento della topologia delle rete di sensori può essere divisa in tre fasi:

• **Pre-deployment phase** è la fase della prima disposizione dei sensori.
Possono essere posizionati in diversi modi: gettati da un aereo, da una catapulta, collocati da un robot o addirittura da un persona umana.
• **Post-deployment phase** è la fase di mantenimento, in questa fase può avvenire un cambiamento della topologia attraverso il movimento dei sensori o il guasto di essi.
• **Re-deployement phase** è la fase di aggiunta di nuovi nodi per sostituire quelli guasti.

1.4.6 Mezzi trasmissivi

La comunicazione senza fili tra i diversi dispositivi in una WSN può essere fatta impiegando sistemi induttivi, ottici o ad infrarossi, ultrasuoni ed infine a radio-frequenza.
Ciascuno di questi mezzi presenta pregi e difetti. In primo luogo, devono essere disponibili in ogni parte del mondo, per evitare problemi di ordine legislativo passando da uno stato all'altro.
Analizziamo con maggior dettaglio i mezzi trasmissivi citati:

• **Sistemi induttivi**:
Questa tecnologia viene principalmente utilizzata nel campo del *Radio Frequency IDentification* (RFID); si utilizzano dei *tag* (marcatori) attivi o passivi che sono letti da apposite porte di lettura. Tali porte generano un forte campo magnetico, in corrispondenza del quale, una induttanza realizzata all'interno del tag manifesta una corrente indotta che permette la lettura/scrittura di informazioni sul tag stesso.

Svantaggi:

- le operazioni di lettura/scrittura possono avvenire correttamente solo su brevi distanze
- i coordinatori della rete (porte o varchi di lettura) richiedono grandi livelli di potenza.

• **Sistemi ottici**:

In questo gruppo includiamo l' ***I**nfra**r**ed **D**ata* A*ssociation* (IrDA) ed altri standard che prevedono l'uso di tecnologie di tipo ottico.

Svantaggi:

- necessità di un collegamento diretto in *line of sight* dei dispositivi, cioè ricevitore e trasmettitore devono essere correttamente allineati.

Vantaggi:

- economicità dei transceiver
- affidabilità

• **Ultrasuoni**:

Vantaggi:

- Consentono di superare la limitazione del line of sight tipica dei sistemi ottici

Svantaggi:

- la generazione di ultrasuoni richiede una elevata energia per il coordinatore della rete; Conseguentemente, questo comporta costi e dimensioni rilevanti per i dispositivi realizzati impiegando tali tecnologie.

• **Radio frequenza (RF)**:

Vantaggi:

- eliminazione dei problemi di line of sight
- lo stato dell'arte nel campo dei transceiver a radio frequenza garantisce dimensioni e consumi molto contenuti.

Svantaggi:

- problema della disponibilità del mezzo trasmissivo: le bande di frequenza vengono assegnate attraverso legislazioni locali in ogni stato. Ciò comporta che la scelta della banda trasmissiva utilizzata debba concentrarsi su bande disponibili nella più vasta gamma di nazioni

1.5 Applicazioni

Le reti di sensori possono essere implementate utilizzando una vasta tipologia di sensori (sismici, magnetici, termici, infrarossi, acustici, radar, ecc.) che sono in grado di monitorare un'ampia classe di condizioni ambientali, tra le quali possiamo ricordare:

- Temperatura;
- Umidità;
- Movimenti di veicoli;
- Condizioni di illuminazione;
- Pressione;
- Livelli di rumore;
- Presenza o assenza di determinati tipi di oggetti;
- Stress meccanici;
- Velocità, direzione e dimensione di oggetti.

Alcune delle principali applicazioni delle reti di sensori *wireless* possono essere classificate in cinque macro gruppi: applicazioni militari, applicazioni industriali, controllo ambientale, applicazioni mediche, *home automation*.

1.5.1 Applicazioni militari

Le reti di sensori *wireless* possono diventare parte integrante delle più comuni attività militari come

- il comando
- il controllo dei campi di battaglia
- la rilevazione degli spostamenti delle truppe nemiche
- la sorveglianza
- le operazioni di localizzazione dei bersagli.
- Il controllo e il rilevamento dello stato degli equipaggiamenti

- riconoscimento di agenti chimico fisici nell'ambito di battaglie chimico biologiche

Questo è reso possibile dall'elevato numero di nodi, dal costo contenuto, che caratterizza una WSN; questi dispositivi possono essere impiegati in grandi quantità anche in ambienti inospitali quale può essere un campo di battaglia.
L'eventuale distruzione infatti di uno o più nodi non influenza l'efficienza della rete, cosa che invece potrebbe accadere utilizzando reti cablate.

1.5.2 Applicazioni industriali

L'utilizzo delle LR-WPAN nel settore industriale s'inserisce nella continua ricerca della diminuzione dei costi per implementare sistemi di controllo per processi produttivi.
I primi impieghi si hanno laddove non sono richiesti elevati *data rate* (velocità di trasmissione) e quindi in applicazioni non critiche per le quali gli intervalli di campionamento non risultano essere un problema.
Al contrario, l'attenzione viene posta sui costi di implementazione e di manutenzione. Ciò comporta la necessità di dispositivi che non richiedano manutenzione, laddove per manutenzione si intende primariamente la necessità di sostituire e/o ricaricare le batterie.

1.5.3 Controllo ambientale

Alcune delle applicazioni delle WSN in questo ambito possono essere:

- sistemi di prevenzione degli incendi
- statistiche relative alla fauna protetta
- agricoltura di precisione
- ricerche meteorologiche e geofisiche
- controllo dell'inquinamento.

Si consideri ad esempio l'impiego di una rete di sensori wireless nella lotta agli incendi. Un numero elevato di sensori può essere posizionato in zone strategiche, in modo casuale all'interno di una vasta area boschiva. L'utilizzo di sensori wireless permette di superare gli ostacoli tipici degli ambienti boschivi quali le rocce, gli alberi e la vegetazione in genere, che non consentirebbero l'installazione delle corrispettive versioni cablate, a meno di considerare interventi radicali molto costosi e distruttivi.

L'agricoltura di precisione

Questo campo di sviluppo prevede l'utilizzo di sensori distribuiti in grado di monitorare alcuni parametri fondamentali per le coltivazioni come:

- le concentrazioni di nitrati
- la temperatura del suolo
- la composizione del terreno
- la quantità di acqua piovana
- l'umidità relativa delle coltivazioni

In questo modo è possibile studiare dei sistemi di controllo che consentano di migliorare l'agricoltura sia in termini di qualità del prodotto finale sia per quanto concerne la quantità, apportando quindi un notevole vantaggio economico per il produttore.
Perché questo tipo di sistema di controllo applicato all'agricoltura possa essere efficace è necessario che l'informazione di ogni sensore sia correlata alla sua posizione affinché l'utente sia informato sulle zone precise in cui intervenire.
Le difficoltà che si possono incontrare nel progetto di questo tipo di rete sono le particolari topologie necessarie, poiché, per coprire vaste aree, sono necessarie reti mesh che consentano cioè ad alcuni nodi di funzionare come ripetitori del segnale inviato da altri, in modo tale che il messaggio possa giungere fino al corretto destinatario.

1.5.4 Applicazioni mediche

Alcuni esempi in questo campo possono essere

- la trasmissione dei parametri fisiologici dei pazienti all'interno degli ospedali
- attività diagnostiche,
- somministrazione di medicinali,
- *personal healthcare*

L'utilizzo di WPAN all'interno delle strutture ospedaliere consente di poter monitorare i parametri fisiologici dei pazienti (temperatura, pressione sanguigna, pulsazioni cardiache) in modo non invasivo per il paziente stesso e consentendo l'intervento tempestivo dei medici in caso di bisogno.
Applicazioni simili possono essere individuate anche nel *personal healthcare*: basti pensare ad una serie di sensori dotati di interfaccia wireless integrati, ad esempio, all'interno di un orologio da polso che consenta la misurazione dei battiti cardiaci, o in una bilancia per monitorare il peso; in questo modo, con una trasmissione giornaliera verso un PDA o un *personal computer,* è possibile costituire un archivio personale in cui vengono memorizzate le informazioni salienti del nostro stato di salute.
Un'altra applicazione interessante è il controllo remoto di persone anziane per prevenire situazioni di pericolo quali ad esempio una caduta o uno sbalzo improvviso delle pulsazioni cardiache.

1.5.5 Home and building automation

Il progetto di una casa intelligente si può esplicare in due diversi approcci progettuali:

- sistema *human centered:* prevede che la tecnologia sia in grado di rispondere alle esigenze dell'utente finale in termini di interazioni input/output.
- sistema *technology centered:* vuole creare un cosiddetto *smart environment*, in cui ogni dispositivo della casa integra uno *smart device* in grado di comunicare con un *server* di stanza. Tale server, a sua volta, è in grado di

comunicare con i server delle stanze adiacenti in modo da creare un sistema integrato autoconfigurante, ed auto organizzato.

- Sistema *Heating, Ventilation and Air Conditioning* (HVAC): in grado di regolare la temperatura di ogni stanza sulla base di diversi sensori disposti, ad esempio, in corrispondenza delle finestre per verificare se sono aperte o chiuse, in corrispondenza delle superfici vetrate in modo da poter valutare l'effetto della radiazione solare sulla temperatura della stanza.

1.6 Architettura dei mote

Nella loro forma più nota, le WSN sono reti formate da tanti piccoli elementi chiamati *mote*, dal nome utilizzato per i primi prototipi di WSN sviluppati dall'Università della California Berkeley (USA). Un mote è un dispositivo di elaborazione in miniatura dotato di memoria, di un apparato ricetrasmittente, di sensori per la rilevazione di dati ambientali e di una batteria per l'alimentazione. I mote vengono progettati in modo che, una volta accesi e inseriti nel luogo di funzionamento, possano essere in grado di operare a lungo senza intervento umano. Tali dispositivi sfruttano le trasmissioni radio per comunicare tra loro e per trasmettere i dati raccolti. L'assenza di cavi di alimentazione e di trasmissione dati facilita il dispiegamento delle WSN, consentendone l'utilizzo in tutte quelle situazioni dove i tradizionali sensori cablati non possono essere impiegati efficacemente.
Un mote di una rete di sensori è descritto dallo schema riportato in Figura 3.

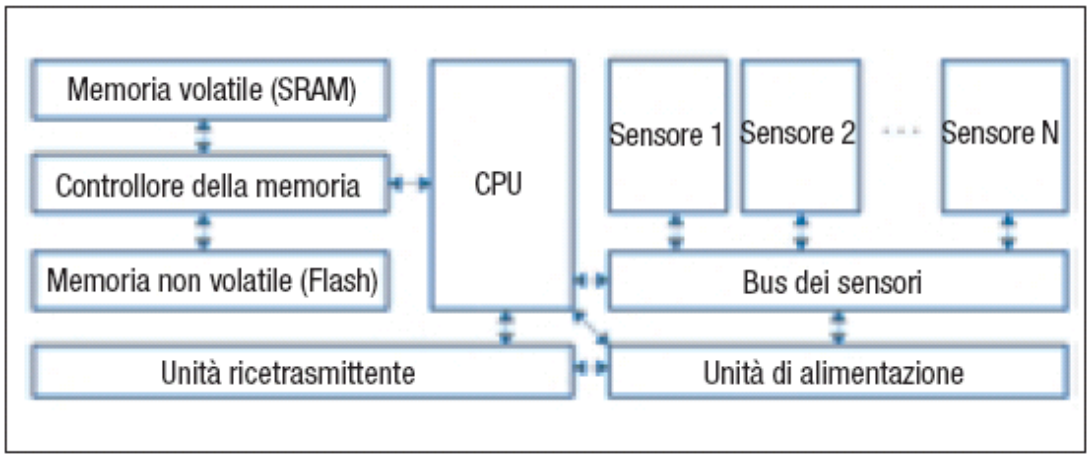


Figura 3: schema hardware di un mote

I componenti principali sono: l'unità di elaborazione (CPU), le memorie, l'unità ricetrasmittente e l'insieme di sensori di cui il mote è corredato.
Nel seguito del paragrafo si analizzeranno con maggior dettaglio i componenti dell'architettura.

1.6.1 Unità di elaborazione

Rispetto ai sensori tradizionali, i componenti di una WSN hanno la capacità di elaborare le informazioni: un mote può rimuovere il rumore di fondo dai segnali raccolti, trasformare i dati raccolti in informazioni di più alto livello e più sintetiche (esempio, il calcolo della media delle temperature rilevate in un periodo) o aggregarle con informazioni ricevute dagli altri mote prima di trasmetterle al destinatario. Queste operazioni sono svolte dall'unità di elaborazione, la quale oltre a elaborare i dati raccolti dai sensori si occupa anche di gestire le risorse di un mote. I microprocessori impiegati devono essere ottimizzati per consumare poca energia, che costituisce una risorsa scarsa nelle normali condizioni di utilizzo, infatti i mote difficilmente possono essere recuperati, una volta dispiegati nell'ambiente di lavoro; per questo motivo prolungare la durata della batteria è importante. Le unità di elaborazione attualmente disponibili sono a 8, 16 e 32 bit. In linea generale si può affermare che i microprocessori a 32 bit offrono elevate prestazioni di calcolo ma consumano più energia; al contrario, i processori a 8 bit sono parchi nel consumo di energia, ma a scapito delle prestazioni.

1.6.2 Memorie

I mote usano memorie SRAM come *memoria di lavoro* e memorie Flash *come memoria di massa*. Le memorie SRAM generalmente sono integrate nel processore; le memorie flash hanno capacità di poche centinaia di kbyte, arrivando ad alcuni Mbyte solo nei mote più dotati. Queste memorie, che hanno la caratteristica di conservare il loro contenuto anche quando non vengono alimentate, sono utilizzate per ospitare sia i programmi sviluppati dagli utenti per i mote, sia i dati rilevati dai sensori in attesa di essere elaborati o trasmessi.

1.6.3 Unità ricetrasmittente

Sono disponibili diverse soluzioni per realizzare il sistema di comunicazione di un mote. Alcuni mote, per esempio i mote *Crossbow* nella serie Mica2, utilizzano dei protocolli di comunicazione proprietari, mentre altri utilizzano protocolli standard come Bluetooth, IEEE 802.15.4/ZigBee e IEEE 802.11 (WiFi). IEEE 802.11 (WiFi) è un protocollo per wireless LAN che risulta carente sotto il profilo del risparmio energetico e per questo motivo è scarsamente utilizzato nelle WSN. Bluetooth e IEEE 802.15.4/ZigBee sono protocolli per WPAN (*Wireless Personal Area Network*).
Le WPAN sono nate per sostituire, con collegamenti radio, i cavi di connessione dei vari dispositivi "*indossati*" da una persona e implementano strategie di risparmio energetico. Ogni mote, poi, mette a disposizione dei programmatori servizi per crittografare i dati trasmessi.

1.6.4 Sensori

I mote hanno sia sensori *built-in* (incorporati) sia un bus di espansione per mezzo del quale è possibile aggiungere altri sensori necessari per svolgere funzioni specifiche. Sono disponibili molti tipi di sensori, per esempio rilevatori di temperatura, rumore,

ricevitori GPS ecc.. È tuttavia possibile costruire un sensore ad hoc specifico per le proprie esigenze.

Un sensore è composto generalmente da un trasduttore e da un convertitore di segnali analogici in digitali. I trasduttori sono costruiti sfruttando le proprietà di certi materiali che variano le loro caratteristiche elettriche al variare delle condizioni ambientali. Un ADC (*Analog to Digital Converter*, convertitore di segnali analogici in segnali digitali) converte il valore di tensione su un trasduttore in un valore binario che verrà poi utilizzato per le successive elaborazioni.

Molti dei trasduttori utilizzati sui mote sono dei MEMS (*MicroElectroMechanical Systems*): si tratta di dispositivi in grado di rilevare una vasta gamma di fenomeni fisici in maniera efficiente ed economica dal punto di vista del consumo energetico; confrontati con i sensori piezoelettrici ad alta precisione, i MEMS forniscono una discreta precisione pur avendo un costo di produzione notevolmente inferiore.

Un MEMS viene costruito utilizzando processi di *etching* (incisione) del silicio (il procedimento è analogo a quello utilizzato per stampare circuiti integrati) per costruire delle piccolissime strutture meccaniche con dimensioni dell'ordine 5 – 10μm. Su queste microstrutture, man mano che vengono costruite, vengono anche stampati dei circuiti a semiconduttore.

La forza di gravità o le accelerazioni possono flettere le strutture di silicio, causando delle variazioni alle caratteristiche elettriche dei circuiti semiconduttori. Il primo esempio di impiego su larga scala di un sensore MEMS è l'accelerometro che causa l'apertura degli airbag nelle automobili.

1.6.5 Considerazioni riassuntive

Nella Tabella 1 è riportato un confronto delle principali caratteristiche hardware di alcuni sensori disponibili in commercio.

Nome piattaforma	IMote	Mica2	MicaZ	Telos B	Stargate
CPU (produttore) Modello Velocità Bit	ARM ARM7TDMI 12 MHz 32 bit	Atmel ATMEGA128 8 MHz 8 bit	Atmel ATMEGA128 8 MHz 8 bit	Texas Instruments MSP430 8 MHz 16 bit	Intel PXA255 400 MHz 32 bit
SRAM (KB)	64	4	4	10	64.000
Flash (KB)	512	128 programmi 512 dati	128 programmi 1024 dati	48 programmi	32.000
Radio	Bluetooth	Proprietario 315/433/915MHz	ZigBee	ZigBee	Espandibile con periferiche di comunicazione via porta PCMCIA, CF, RS 232, USB, Ethernet
Banda (kbit/s)	720	15	250	250	

Tabella 1: confronto tra diversi hardware per WSN

1.7 Architettura software

Un mote è un elaboratore,dotato di risorse hardware molto limitate, che deve gestire un apparato ricetrasmittente ed il routing dei pacchetti dati. Il sistema operativo e le applicazioni sviluppate devono essere in grado di esercitare un controllo fine delle risorse hardware per poter implementare politiche di risparmio energetico efficaci che consentano di far durare il più possibile la batteria che alimenta il mote. I sistemi operativi tradizionali (come Linux, OSE, QNX, . . .) impiegati nei sistemi embedded sono carenti per quanto riguarda i requisiti appena introdotti; per questo sono stati sviluppati dei sistemi operativi ad hoc come TinyOS [2], EYES OS e Contiki.

L'implementazione delle politiche di risparmio energetico e di routing delle informazioni costituiscono un fattore cruciale di una WSN. Il risparmio di energia può essere ottenuto spegnendo i sensori di rilevazione quando non vengono utilizzati, *addormentando* l'unità di elaborazione quando non deve svolgere calcoli e, soprattutto, usando oculatamente l'unità ricetrasmittente, il componente che in assoluto consuma più energia. Un mote può decidere di spegnere l'unità ricevente per brevi intervalli di tempo, può sfruttare la comunicazione *multihop*[1] e, soprattutto, può cercare di elaborare

localmente le informazioni prima di trasmetterle. Un sensore che deve calcolare la temperatura media di un ambiente in un giorno, può memorizzare i valori man mano che vengono raccolti e a fine giornata calcolare e trasmettere la media. In questo caso il consumo energetico per memorizzare i valori, elaborarli e trasmetterli solo a fine giornata è notevolmente inferiore rispetto a quanto consumerebbe l'apparato di trasmissione per trasmettere singolarmente ognuno dei valori rilevati.

1.7.1 TinyOS e NesC

Analizziamo ora NesC e TinyOS, che sono, rispettivamente, il primo linguaggio utilizzato per scrivere le applicazioni per mote ed il primo sistema operativo per mote (scritto in NesC), entrambi sviluppati presso l'Università di Berkeley. TinyOS è diventato uno standard industriale, sia per essere stato il primo sistema operativo per WSN, sia perché i suoi sorgenti sono *open source* (pubblici e liberamente utilizzabili).

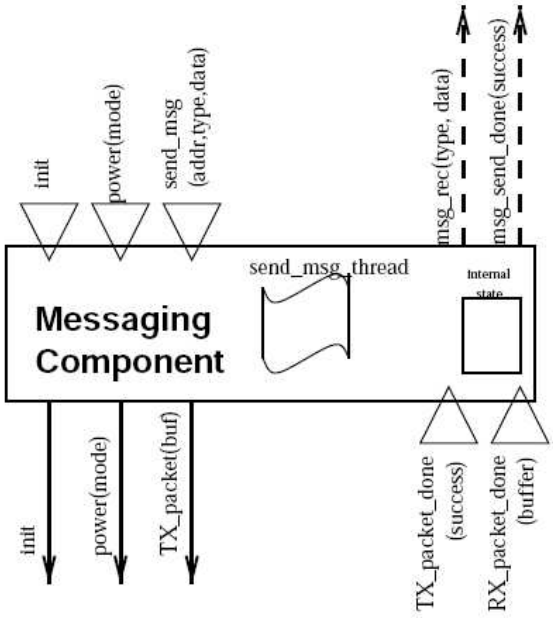


Figura 4: rappresentazione di un componente di TinyOS

I mote sono dei dispositivi di elaborazione particolari, perché, per la maggior parte del tempo, sono in uno stato di *sleep* (dormono). L'elaborazione svolta è guidata dagli eventi: solo quando il sensore acquisisce nuovi dati oppure quando vengono ricevuti dei

messaggi si avvia un processo di calcolo. L'unità base di programmazione e di compilazione nel linguaggio NesC è il *componente;* ogni programma è ottenuto assemblando uno o più componenti tra loro. Ogni componente è specificato da una o più interfacce, le quali, in modo simile a quanto fatto da altri linguaggi, dichiarano un insieme di funzioni e procedure da implementare chiamate *comandi*. A differenza di linguaggi come C e JAVA, le interfacce NesC offrono dei costrutti sintattici espliciti sia per specificare il codice di gestione degli eventi (la porzione di codice che deve essere avviata al verificarsi di un particolare tipo di evento) sia per gestire l'accoppiamento tra i componenti che producono eventi e i componenti interessati alla notifica.

I componenti NesC a loro volta si dividono in due categorie: i *moduli* e le *configurazioni*. I moduli contengono il codice che implementa una o più interfacce; un componente di configurazione (più brevemente, configurazione) viene utilizzato invece per specificare come assemblare più moduli tra loro. L'obiettivo principale di quest'approccio è poter sviluppare separatamente componenti che possono essere poi facilmente composti tra loro. TinyOS è formato da un insieme di *routine* (procedure) di sistema, implementate per mezzo di componenti NesC, e da un componente che si occupa di pianificare l'esecuzione dei *thread* (Parti atomiche di un processo). In un mote ci sono solamente due thread in esecuzione: l'esecutore dei *task* e l'esecutore dei *gestori degli eventi hardware*. I *task* sono blocchi di codice applicativo, i gestori degli eventi hardware si occupano di gestire gli *interrupt* (interruzioni) generati dall'hardware del mote, in particolare le interruzioni generate dai sensori e dall'unità ricetrasmittente.

Le chiamate di sistema TinyOS che richiedono lunghe elaborazioni in termini di tempo non sono bloccanti, ma vengono eseguite in modo asincrono: la loro invocazione restituisce subito il controllo al chiamante ed il completamento del comando è segnalato per mezzo di un evento. Per esempio, l'interfaccia che offre il comando per trasmettere dati via radio richiede al chiamante di gestire l'evento "send-Done", generato dal sistema operativo, che informa che il pacchetto è stato inviato.

1.7.2 Macchine virtuali

I programmi per WSN vengono sviluppati su desktop, provati con dei simulatori e poi trasferiti sui mote. Per quest'ultima operazione si utilizza la *programming board,* una scheda che si collega ad una porta del PC, sulla quale si può inserire un mote e che permette di scrivere direttamente sulla memoria flash del mote stesso. Con questo metodo è necessario riprogrammare i mote uno ad uno ed inoltre non è possibile riprogrammare quelli già disposti sul campo. Sarebbe auspicabile utilizzare le capacità di comunicazione dei mote per distribuire un nuovo programma, tuttavia la maggior parte dell'hardware non permette di riscrivere via radio la memoria delle applicazioni.
Questo inconveniente può essere superato eseguendo su ogni mote una *virtual machine* (macchina virtuale), un interprete di codice macchina che gira come un normale programma.
La macchina virtuale esegue del codice che può essere trasmesso come se fosse una normale sequenza dati, quindi sfruttando le potenzialità di comunicazione di una rete di mote. Le istruzioni interpretate dalle macchine virtuali sono generalmente istruzioni di più alto livello di quelle offerte dall'hardware e da TinyOS, per questo motivo il codice prodotto ha dimensioni inferiori ed è quindi più facile da trasmettere.
A differenza di un programma NesC, il quale viene eseguito direttamente dall'hardware del mote senza alcun controllo, la macchina virtuale può controllare che il programma interpretato non esegua operazioni pericolose che possano portare a bloccare il mote, come scrivere in zone di memoria al di fuori di quelle assegnate.
Esempi di applicazioni o sistemi operativi che implementano al loro interno delle macchine virtuali sono: Mate, Bombilla, Sensorware, MagnetOS.

1.7.3 TinyDB

I mote di una WSN hanno la capacità di acquisire informazioni autonomamente dall'ambiente esterno e hanno capacità di elaborazione che possono essere utilizzate per filtrare ed in generale per pre-elaborare localmente i dati raccolti.

Queste capacità sono state sfruttate in modo interessante da TinyDB[1][3], un software che permette di creare una rappresentazione astratta di un insieme di mote basata sul paradigma del database. TinyDB permette di recuperare informazioni eseguendo interrogazioni espresse in linguaggio SQL come se ci si trovasse di fronte ad un database vero e proprio.

L'architettura software sottostante a TinyDB si occupa di inviare la *query* (interrogazione) ai mote interessati, di farla eseguire in locale, di raccogliere le informazioni e di presentarle all'interrogatore, eventualmente in forma aggregata. Essa si occupa anche di minimizzare il traffico di rete, per esempio cercando di inviare le query solamente ai nodi che dispongono delle informazioni cercate e aggregando i dati raccolti dai nodi vicini prima di ritrasmetterli. Il linguaggio di TinyDB include anche un paradigma attivo, che prevede la possibilità di scrivere query che vengono scatenate periodicamente o in presenza di determinati eventi.

Il prossimo capitolo sarà completamente dedicato alla gestione delle interrogazioni e della raccolta dati effettuate da TinyDB.

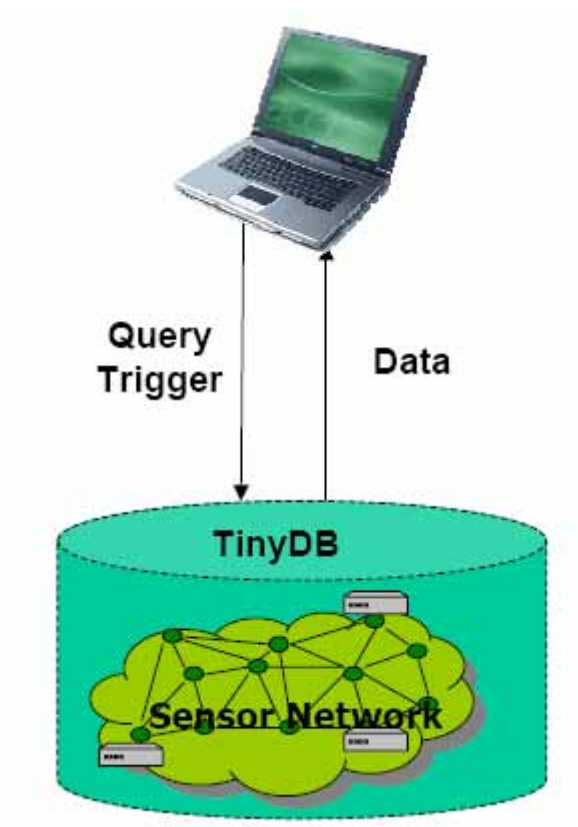
Capitolo 2

TINYDB

2.1 Introduzione a TinyDB

TinyDB è un sistema di elaborazione per l'estrazione delle informazioni ottenute da una rete di sensori TinyOS.

A differenza delle altre soluzioni per l'acquisizione dei dati offerte dal sistema, con TinyDB l'utente non ha più l'onere di dover programmare personalmente ogni sensore tramite la scrittura di centinaia di linee di codice in linguaggio C. Al contrario, fornisce una semplice interfaccia in un linguaggio simile all'SQL (SQL-like) che consente di specificare i dati da acquisire e, a differenza del linguaggio tradizionale, parametri aggiuntivi come il tasso con cui aggiornare i dati.



Definita una query e specificati gli attributi d'interesse, TinyDB raccoglie i dati dai mote, li filtra, li aggrega insieme e li inoltra a un PC. Per fare questo sfrutta particolari algoritmi di elaborazione dei dati all'interno della rete, grazie ai quali è in grado di ridurre al minimo il consumo di energia, fattore critico per le reti di sensori wireless.

TinyDB fornisce una semplice API Java[6] per la scrittura di applicazioni che consentono di interrogare la rete ed estrarre dati dai sensori.

TinyDB nasce essenzialmente per semplificare il compito del programmatore, liberandolo dall'onere di dover scrivere numerose linee in codice di basso livello per ogni dispositivo.
Il sistema include:

- *Metadata Management* (Gestione dei metadati) TinyDB fornisce un elenco di metadati per descrivere i diversi tipi di rilevamenti che è possibile effettuare nella rete.

- *High level* query (Interrogazioni di alto livello) TinyDB usa un linguaggio dichiarativo che permette all'utente di specificare i dati d'interesse, senza dover tuttavia indicare come ottenerli.

- *Network Topology* (Topologia della rete) TinyDB gestisce la rete sottostante mantenendo tabelle di routing ed assicurando che ogni mote possa distribuire i propri dati in modo efficiente e relativamente affidabile.

- *Multiple* Query (interrogazioni multiple) TinyDB consente l'esecuzione simultanea di più query sullo stesso set di nodi. Le interrogazioni possono avere diversi tassi di campionamento e accedere a tipi diversi di sensori.

- *Incremental Deployment via* Query *sharing:* per espandere la rete è sufficiente installare il codice di TinyDB sui nuovi motes, senza doversi occupare della loro configurazione o programmazione. Le interrogazioni sono distribuite su tutti i nodi della rete: se un dispositivo rileva nel canale radio la presenza di un messaggio per una query che non ha ancora eseguito, automaticamente richiede al mittente una copia dell'interrogazione ed inizia ad eseguirla.

Il sistema è eseguito su ogni singolo nodo della rete ed è composto da due macrosezioni: il software eseguito sui singoli sensori, sviluppato in nesC, e l'interfaccia client, sviluppata in Java.
Il tipico processo di funzionamento inizia quando il client invia una query al nodo collegato fisicamente ad esso (tipicamente un mote collegato via cavo a un computer).
La query a questo punto viene diffusa a tutti i sensori della rete costruendo una struttura di routing ad albero che ha il nodo radice collegato a un computer.
Ogni nodo rileva i dati dai propri sensori, raccoglie le eventuali informazioni che arrivano dai nodi a valle, se necessario, le aggrega e infine propaga tali informazioni ai nodi a monte. Quando tutte le informazioni sono giunte al nodo radice, la query termina e viene visualizzato il risultato all'utente o all'interfaccia Java.

Il software contenuto sul singolo sensore è logicamente composto da diversi componenti[6]:
• *Sensor catalog* Questo componente tiene traccia del set delle misure disponibili, e di alcune proprietà che caratterizzano il singolo sensore (ad esempio il sensore padre nella struttura ad albero usata per il routing).

• Query *processor (ottimizzatore)* Questo è il componente chiave durante l'esecuzione di una query; infatti esso riceve i dati rilevati dall'hardware e dai sensori figli, aggrega e filtra tali dati e li rispedisce al sensore padre a intervalli di tempo specificati nella query stessa.

• *Network topology manager* Questo componente fornisce un'interfaccia per l'interazione con altri componenti di un livello sottostante per le funzioni di routing. Fornisce i comandi per inviare query, i risultati e gli eventi e per ricevere query e risultati.

L'interazione da parte dell'utilizzatore della rete avviene mediante un'interfaccia grafica se l'utilizzatore è un essere umano o attraverso una serie di API Java concettualmente non molto diverse da JDBC.
Il concetto di query è lievemente diverso da quello dei DBMS tradizionali:

in TinyDB, la query viene iniettata nella rete e rimane *attiva* per un tempo specificato. Durante questo periodo, i sensori ritrasmettono i loro dati a intervalli di tempo precisi, per cui una query aggiorna i dati in possesso dell'utilizzatore a intervalli costanti invece che una sola volta.
Il sistema è sviluppato per sensori mote sotto forma di applicazione TinyOS sviluppata in nesC.

2.2 Sviluppatori

Il database distribuito TinyDB è stato sviluppato presso l' Intel Research Laboratory Berkeley e UC Berkeley per reti wireless di piccoli sensori chiamati &quotmotes.
Al progetto hanno collaborato lo scienziato, ricercatore dell'Intel, Wei Hong, insieme a Samuel Madden, Michael J. Franklin, and Joseph M. Hellerstein dell'UC Berkeley.

Team di ricerca

Eric Brewer, UC Berkeley
David Culler, Intel Research Berkeley lab director and UC Berkeley professor
Michael J. Franklin, UC Berkeley
David Gay, Intel Research scientist
Joseph M. Hellerstein, UC Berkeley
Wei Hong, Intel Research scientist
Phil Levis, UC Berkeley
Samuel Madden, UC Berkeley
Robert von Behren, UC Berkeley
Matt Welsh, Intel Research scientist

2.3 Linguaggio di acquisizione

In questo paragrafo si presenterà il linguaggio, SQL-like, messo a disposizione da TinyDB per la gestione delle interrogazioni all'interno della rete. In particolare,

l'attenzione sarà focalizzata sulle questioni relative al momento in cui iniziare il campionamento dei dati e alla frequenza con cui condurre tale operazione, sino al termine della query.

2.3.1 Modello dei dati

In TinyDB le tuple, acquisite da ogni nodo della rete, sono inserite nella medesima tabella, ***sensors***. Tale schema presenta una colonna per ogni attributo che il dispositivo può fornire (e.g. luce, temperatura, umidità, ecc). Ciascun record, contenente i dati campionati da un singolo nodo in un dato istante di tempo, è materializzato all'interno della tabella solo nel caso in cui i suoi parametri soddisfino la richiesta della query. Le tuple restano in memoria per un periodo limitato, sino al termine della loro elaborazione.

Dal momento che il sistema impone la medesima tabella per qualsiasi nodo della rete, è necessario tenere conto del fatto che non tutti i motes potrebbero essere in grado di fornire i dati per ciascun attributo. In tal caso, al dispositivo è data la possibilità di inserire il valore di default NULL per i valori mancanti. La clausola WHERE all'interno dell'interrogazione esclude automaticamente le tuple con tale dato.

Da un punto di vista fisico la tabella *sensors* è distribuita su tutta la rete, all'interno della quale ogni dispositivo campiona e salva i dati letti. Il confronto di tali informazioni, campionate in uno stesso istante ma provenienti da nodi differenti, può essere condotto solo a seguito della raccolta di tali dati in alcuni nodi comuni (e.g. il nodo radice).

2.3.2 Caratteristiche del linguaggio

In TinyDB, così come in SQL, le interrogazioni sono eseguite tramite l'utilizzo delle istruzioni SELECT, FROM, WHERE, GROUP BY, con le quali è possibile realizzare le operazioni di selezione, proiezione, join e aggregazione. L'impiego di tali clausole è il medesimo.

L'istruzione FROM può riferirsi alla tabella *sensors*, ma anche alle tabelle salvate localmente, dette punti di materializzazione (*materialization points*). Questi ultimi sono creati tramite apposite query e saranno descritti nel corso del paragrafo.

Le tuple sono prodotte a ben definiti intervalli di campionamento (*sample interval*), indicati come parametri della stessa query.
Si definisce *epoch* il periodo di tempo che intercorre tra due istanti di campionamento successivi.

Consideriamo la seguente query:

```
SELECT nodeid, light, temp
    FROM sensors
    SAMPLE PERIOD 1s FOR 10s
```

Con questa interrogazione si chiede ad ogni dispositivo (appartenente alla tabella virtuale *sensors*) di riportare il proprio identificativo (nodeid), la luminosità e la temperatura rilevate, effettuando una lettura ogni secondo, per un periodo totale di 10 secondi.
I risultati, tramite una topologia di trasmissione *multihop*, risalgono tutta la rete sino alla radice, nella quale saranno registrati o visualizzati all'utente. Tale output consiste in uno *stream* (flusso) di tuple, lette a intervalli di tempo di un secondo.
Ogni record, oltre agli attributi relativi ai dati campionati, presenta anche una chiave cronologica (*timestamp*), indicante il momento in cui è stato fornito il risultato.

TinyDB sfrutta un semplice protocollo di sincronizzazione grazie al quale, accordandosi con la base tempi globale, consente ai lettori di iniziare e terminare ogni *epoch* nello stesso istante.

Ad ogni query, nel momento della sua emissione, viene assegnato un identificatore (id), grazie al quale l'utente potrà, ad esempio, fermare l'interrogazione, tramite il comando"STOP QUERY id".

Metodi alternativi per terminare l'esecuzione di una query sono l'utilizzo dell'istruzione FOR, vista nell'esempio precedente, oppure la definizione di una condizione di stop.
Si ricorda che, a differenza di quanto accade nei database tradizionali, la tabella *sensors* è costituita da un continuo e illimitato flusso di valori. Diventa quindi impossibile eseguire su di essa operazioni di join o di ordinamento.
Tale operazione diventa tuttavia possibile se si definisce un sottoinsieme limitato (*window*) dello *stream* di valori. Le finestre, in TinyDB, sono definite attraverso la creazione di *materialization point* (punti di materializzazione). Questi costituiscono tabelle, di lunghezza limitata, salvate nella memoria locale di alcuni nodi e contenenti un certo numero di tuple. Il salvataggio ne consente il successivo riutilizzo in altre query.

```
CREATE
    STORAGE POINT recentlight SIZE 8
    AS (SELECT nodeid, light FROM sensors
    SAMPLE PERIOD 10s)
```

Questa istruzione permette di salvare in una locazione di memoria (ad esempio in un singolo nodo) gli otto valori di luminosità più recenti (e gli identificatori dei rispettivi nodi presso i quali è avvenuto il campionamento), letti dalla tabella *sensors*, a intervalli di 10 secondi.

L'operazione di join sarà consentita tra due M.P.(*Materialization Point*) appartenenti allo stesso nodo oppure tra un M.P. e la tabella *sensors*, utilizzando quest'ultima come relazione più esterna in un nested loop join.

```
SELECT COUNT(*)
    FROM sensors AS s, recentlights AS rl
    WHERE rl.nodeid = s.nodeid
    AND s.light < rl.light
    SAMPLE PERIOD 10s
```

All' arrivo, ogni nuova tupla della tabella *sensors* viene messa in join con le tuple del *Materialization Point* aventi lo stesso identificativo del nodo.
La query fornisce in uscita il numero delle recenti letture memorizzate (da 0 a 8 campionamenti effettuati nel passato), per le quali il valore di luminosità è maggiore di quello corrente campionato allo stesso nodo. Ogni nuova lettura viene effettuata in un arco temporale di 10 secondi.

2.3.3 Query di aggregazione

Nel contesto di una rete di sensori wireless, al fine di migliorare le comunicazioni e minimizzare i consumi, l'aggregazione dei dati si presenta come una possibilità molto attraente che consente di ridurre la quantità delle informazioni da trasmettere.
L'approccio di base seguito da TinyDB consiste nella progressiva aggregazione dei dati all'interno della rete, man mano che questi risalgono verso la radice, in rapporto alla funzione di aggregazione e agli attributi di raggruppamento specificati nella query.

Sintassi e semantica

Iniziamo considerando il caso di un utente interessato al monitoraggio della stanze di riunione situate ad un certo piano di un edificio. In particolare, egli desidera sapere quali stanze sono occupate e, per fare questo, sfrutta dei sensori microfonici inseriti in motes. Tali sensori gli permettono di individuare i locali nei quali il volume medio registrato (assumendo che ogni stanza contenga più sensori) supera una certa soglia.

```
SELECT AVG(volume), room FROM sensors
        WHERE floor = 6
        GROUP BY room
        HAVING AVG(volume) > threshold
        SAMPLE PERIOD 30s
```

La query classifica i nodi della rete in base alla stanza in cui sono posizionati e, successivamente, restituisce in uscita tutte le stanze nelle quali è stato registrato un suono medio superiore alla soglia. I dati sono aggiornati ogni 30 secondi.

Ricordando che le query eseguite su TinyDB forniscono come output uno *stream* di dati, invece di un singolo valore aggregato come accade in SQL, per tali interrogazioni ogni record è definito dalla seguente coppia di valori per gruppo:

< groupid, aggregate value >

Ogni raggruppamento è caratterizzato da un *epoch number:* tutte le tuple utilizzate per il calcolo del valore aggregato del record appartengono a una stessa *epoch.*

Struttura delle aggregazioni

L'approccio seguito da TinyDB consiste nell'implementare l'aggregazione attraverso un *partial state record* (record di stato parziale) e tre funzioni:
- una di fusione *f:* specifica come calcolare un'aggregazione intermedia a partire da altre
- una di inizializzazione *i:* specifica come istanziare un record di stato parziale per un singolo valore
- una di valutazione *e*: calcola il valore di un aggregazione a partire da un record di stato parziale

In generale f ha la seguente struttura:

$$< z > = f(< x >, < y >)$$

Dove < x > e < y > sono record di stato parziale e rappresentano uno stato intermedio calcolato su quei valori che saranno richiesti per calcolare l'aggregazione,

< z > *Partial-state record* risultante dall'applicazione della funzione f su < x > e < y >

Esempio:

- *f* è la funzione di fusione per AVERAGE
- ogni record di stato parziale è costituito dalla coppia di valori : SUM e COUNT: < S, C >.

Per cui, dati due record di stato parziale < S1, C1 > e < S2, C2 >,
la funzione f è definita come:

$$f(< S1, C1 > , < S2, C2 >) = < S1 + S2, C1 + C2 >$$

La funzione d'inizializzazione *i* specifica come istanziare uno state record per un singolo valore:
per un AVERAGE calcolata sul sensore *x*, *i(x)* ritorna la tupla < *x*, 1 >.
Infine, la funzione di valutazione *e* calcola il valore attuale dell'aggregazione a partire da un partial state record.
In questo caso, per la media, la funzione *e* è:

$$e(< S, C >) = S/C$$

2.3.4 Aggregazioni temporali

Oltre alle aggregazioni calcolate su valori campionati nello stesso intervallo di lettura, l'utente potrebbe volere la possibilità di eseguire operazioni temporali.

Riprendendo l'esempio del monitoraggio delle sale di un edificio, l'utente può decidere di monitorare l'occupazione delle stanze valutando il volume medio registrato all'interno di ognuna per un certo arco di tempo e riportando periodicamente tale valore.

Ad esempio, la query

```
SELECT WINAVG(volume, 30s, 5s)
        FROM sensors
        SAMPLE PERIOD 1s
```

Riporta ogni 5 secondi il valore medio del volume misurato negli ultimi 30s, campionando i valori ogni secondo.

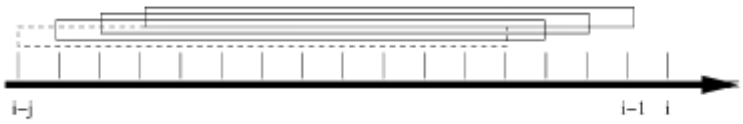


Figura 5: Streaming Window su un flusso di dati

Questo è un esempio di *streaming-window;* sarebbe stato equivalente eseguire una query di aggregazione con un SAMPLE PERIOD di 5s su un *materialization point* di 30 s.

2.3.5 Interrogazioni basate sugli eventi

In alternativa al continuo meccanismo di polling per l'acquisizione dei dati, il sistema consente di condizionare l'esecuzione delle query in base al verificarsi di particolari *events* (eventi). In TinyDB tali eventi sono creati esplicitamente da un'altra query oppure da un componente di basso livello del sistema operativo. In questo caso il codice che genera l'evento deve essere stato precedentemente compilato all'interno del nodo.

Ad esempio, la query:

```
ON EVENT bird-detect(loc):
        SELECT AVG(light), AVG(temp), event.loc
        FROM sensors AS s
        WHERE dist(s.loc, event.loc) < 10m
```

SAMPLE PERIOD 2s FOR 30s

può essere impiegata per riportare il livello medio di luce e temperatura letto dai sensori presenti in prossimità del nido presso il quale è stato rilevato un uccellino. Ogni volta che si verifica l'evento *bird-detect,* il nodo che lo rileva esegue la query e, per 30 secondi, vengono raccolte la temperatura e la luce media dai nodi circostanti, effettuando una lettura ogni 2s.

Tali eventi sono di centrale importanza nel processo di acquisizione poiché consentono al sistema di restare inattivo, in uno stato di *sleep*, per un lungo periodo, sino al momento di occorrenza di una condizione esterna. Questa situazione è da preferirsi a quella di una continua attività di polling che risveglia periodicamente il sistema col fine di verificare l'arrivo di nuovi dati, essendo quest'ultima sicuramente molto più onerosa da un punto di vista energetico.

Nella figura che segue si può vedere la differenza dell'energia consumata nei due casi:

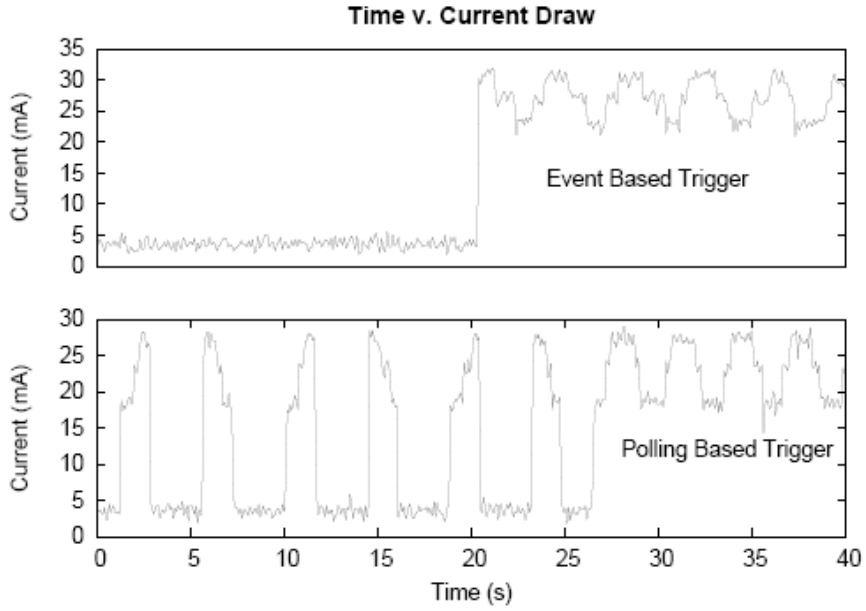


Figura 6: Confronto tra il consumo di potenza in caso di event based trigger e polling based trigger

La parte superiore mostra la corrente che scorre all'interno di un nodo sul quale è eseguita una query basata sugli eventi. L'esecuzione è innescata da un interruttore collegato a una linea di interrupt esterna che provoca la riattivazione del dispositivo.

Nella parte inferiore vediamo invece l'andamento della corrente in un dispositivo in cui l'evento innescante la query è causato da una seconda query che controlla, con una certa frequenza, lo stato di una condizione.

Si può notare facilmente come, nel secondo caso, il consumo di potenza sia nettamente superiore.

Gli eventi possono inoltre agire come condizioni bloccanti per una query:
inserendo la seguente istruzione,

STOP ON EVENT(param) WHERE cond(param)
L'occorrenza dell'evento e l'occorrenza della condizione provocheranno l'interruzione della query.
Oltre alle API di basso livello che consentono ai componenti software di segnalare degli eventi, anche le query stesse possono svolgere questa funzione.

Ad esempio, se si desidera segnalare un evento ogni volta che la temperatura supera una certa soglia, la query segnalatrice dell'evento può essere la seguente:

```
SELECT nodeid, temp
        WHERE temp > tresh
        OUTPUT ACTION SIGNAL hot(nodeid, temp)
        SAMPLE PERIOD 10s
```

In questo modo si perdono ovviamente i benefici discussi precedentemente in merito al consumo di potenza.
Nella versione di TinyDB studiata, gli eventi sono segnalati solo nel nodo locale e la query, sviluppata a partire da tale nodo, viene disseminata verso tutti gli altri.

2.3.6 Interrogazioni basate sul tempo di vita

Abbiamo visto come l'istruzione "SAMPLE PERIOD Xs" consenta all'utente di impostare un tasso di campionamento costante, in base al quale saranno effettuate le letture.
Al contrario, soprattutto nelle applicazioni di monitoraggio ambientale, gli scienziati potrebbero nutrire maggiore interesse nel tempo di vita della rete, piuttosto che in piccole variazioni del tasso di campionamento.
A tal proposito, TinyDB mette a disposizione l'istruzione QUERY LIFETIME < x >, dove < x > esprime la durata minima in giorni, settimane o mesi, richiesta alla rete.

Ad esempio, la query

```
SELECT nodeid, accel
    FROM sensors
    LIFETIME 30 days
```

Richiede il funzionamento della rete per un periodo minimo di 30 giorni, durante il quale i dati dovranno essere campionati con un tasso che sia il più alto possibile, tale però da consentire il raggiungimento dell'obiettivo sul tempo di vita.

TinyDB è in grado di raggiungere tale scopo, calcolando il tasso di campionamento e quello trasmissivo, utilizzando l'informazione sul livello di energia residua, espressa in joule.

Per vedere più in dettaglio come avviene tale stima, consideriamo inizialmente il caso di un singolo nodo, posto alla radice della rete.
Assumiamo inoltre che il tasso di campionamento sia uguale a quello di trasmissione.
In un singolo mote, tali tassi possono essere calcolati grazie a una semplice formula basata sui costi che comprende: costo di accesso ai sensori, selezione degli operatori, tasso atteso di comunicazione, voltaggio della batteria.

Consideriamo, ad esempio, la query:

SELECT $a_1, \ldots, a_{num\mathit{sensors}}$
FROM *sensors*
WHERE *p*
LIFETIME *l* hours

Per semplificare l'analisi, ipotizziamo che l'interrogazione presenti un singolo predicato di selezione, applicato dopo l'acquisizione degli attributi.
Di seguito sono riportati i parametri che saranno utilizzati per il calcolo:

l: tempo di vita desiderato (ore)
c_{rem} : energia rimanente (Joules)
E_n: energia spesa per il campionamento del sensore n (Joules)
E_{trans}: energia spesa per la trasmissione di un singolo campione (Joules)
E_{rcv}: energia spesa nella ricezione del messaggio (Joules)
σ: grado di selettività del predicato
C: numero di figli del nodo considerato

Il primo passo consiste nel determinare la potenza disponibile per ora:

$$p_h = c_{rem} / l$$

Si deve quindi calcolare l'energia necessaria per l'acquisizione e la trasmissione di un singolo campione, includendo i costi derivanti dallo scambio di messaggi coi figli.

$$e_s = (\sum^{num\mathit{Sensors}} Es) + (E_{rcv} + E_{trans}) \times C + E_{trans} \times \sigma$$

l'energia per un singolo campionamento è quindi data dalla somma delle seguenti componenti di energia:

- energia spesa nel campionamento dei sensori presenti nel nodo
- energia spesa per l'acquisizione dei risultati da parte dei noi figli
- energia spesa per l'inoltro dei risultati che soddisfano il predicato della query.

Infine, possiamo quindi calcolare il massimo tasso di trasmissione, T, espresso in campionamenti per ora.

$$T = p_h / e_s$$

Dopo aver considerato il caso di un singolo nodo, procediamo con l'argomentazione discutendo come avvenga il coordinamento del tasso di trasmissione, attraverso tutti i nodi dell'albero.

Dal momento che, come anticipato precedentemente, i dispositivi restano inattivi sino al verificarsi di una particolare condizione, è necessario riuscire a sincronizzare i periodi di risveglio di mittenti e destinatari. Per fare questo, si fa in modo che i nodi possano trasmettere solo quando i padri sono svegli e in ascolto. Ovviamente questa necessità limita il massimo valore del tasso di trasmissione a quello della radice della rete: se un nodo deve trasmettere più lentamente per rispettare la condizione imposta sul tempo di vita, potrà trasmettere a un tasso che sarà un sottomultiplo di quello della radice. Per propagare tale valore attraverso la rete, ogni nodo padre, compreso il nodo *root*, inserisce il proprio tasso di trasmissione nelle query che inoltrerà ai nodi figli.

Come si può notare, tale meccanismo non consente alcun intervento da parte dell'utente; questo potrebbe rappresentare un problema per quelle applicazioni che richiedono un certo grado di precisione nel monitoraggio dei fenomeni fisici.

Gli sviluppatori di TinyDB hanno quindi introdotto un'ulteriore istruzione, "MIN SAMPLE RATE r", che consente all'utente di fissare una frequenza minima di campionamento, al di sotto della quale il sistema non può andare.

Se il tasso minimo imposto è superiore a quello calcolato dal sistema per adeguarsi al requisito imposto sul tempo di vita (LIFETIME), allora, per poter soddisfare entrambe le condizioni, la rete potrebbe essere obbligata ad aggregare o eliminare alcuni dati. Tale situazione sarà discussa nei prossimi paragrafi.

2.3.7 Interrogazioni nelle reti di sensori

Concludiamo la sezione descrivendo brevemente le tipologie di query messe a disposizione da TinyDB:

- ***Monitoring* query** Interrogano la rete in modo continuativo e periodico per ottenere informazioni sul valore di uno o più attributi.

- ***Network health* query** Interrogazioni eseguite direttamente sui nodi della rete per verificarne lo stato. Un esempio può essere rappresentato dalla selezione dei nodi (padri e figli) che presentano un livello di energia residua inferiore a un certo valore. Come si può intuire, queste query sono particolarmente importanti per la natura dinamica e volatile della rete.

```
SELECT nodeid, voltage
        FROM sensors
        WHERE voltage < k
        SAMPLE PERIOD 10 minutes
```

- ***Exploratory* Query** Sono eseguite un'unica volta con lo scopo di analizzare lo stato di un nodo o di un set di nodi in un momento fissato:

```
SELECT light, temp, volume
        FROM sensors
        WHERE nodeid = 5
        ONCE
```

- ***Nested* query**: sia gli eventi quanto i punti di materializzazione costituiscono una forma di query annidate. La versione studiata di TinyDB non supporta questo tipo di interrogazioni: infatti non si può stabilire con chiarezza il momento in cui deve essere effettuata la valutazione della query più esterna, dal momento che quella interna potrebbe fornire in uscita uno *stream* di valori in continuo accrescimento. Al contrario, le query eseguite su punti di materializzazione forniscono all'utente un numero definito di tuple. Utilizzando la clausola FOR, è possibile creare un MP che

contenga un numero limitato di valori, sui quali è possibile eseguibile una query annidata.

- **Actuation query:** consentono all'utente di definire l'attuazione di un'azione (fisica) in risposta all'esecuzione della query. Ad esempio, nel monitoraggio di un edificio, si può essere interessati all'accensione di un interruttore quando la temperatura scende sotto a un certo livello.

  SELECT nodeid, temp
  FROM *sensors*
  WHERE temp > treshold
  OUTPUT ACTION power-on(nodeid)
  SEMPLE PERIOD 10s

  L' istruzione "OUTPUT ACTION" provoca l'invocazione di un comando esterno in risposta a una tupla che soddisfa la query. Inoltre tale clausola interrompe la trasmissione dei messaggi alla stazione base.
- ***Offline delivery*:** ci sono casi in cui l'utente potrebbe essere interessato a registrare fenomeni che si verificano con una frequenza superiore a quella con cui i dati possono essere trasmessi. TinyDB supporta il *log* (registrazione) dei risultati in memorie EEPROM per una rilevazione dei dati non in tempo reale. Questa funzionalità è implementata tramite l'utilizzo dei *materialization point* precedentemente descritti.

2.4 Ottimizzazione basata sui costi

2.4.1 Gestione dei metadati

Ogni nodo in TinyDB mantiene un catalogo di metadati che descrivono gli attributi locali, gli eventi e le funzioni definite dall'utente. Sono copiati periodicamente alla radice della rete per essere utilizzati dal *query processor* (ottimizzatore delle query) e vengono registrati nel sistema mediante link statico effettuato durante la compilazione,

utilizzando il linguaggio NesC di TinyOS. Gli eventi e gli attributi appartenenti ai vari componenti di TinyDB e del sistema operativo sono resi disponibili per le interrogazioni dichiarandoli in un file d'interfaccia e fornendo una piccola funzione per la gestione.

Ad esempio, per consentire all'ottimizzatore delle query di conoscere la topologia della rete, il componente *Network* di TinyOS definisce l'attributo *parent* di tipo integer e registra un gestore di eventi in grado di fornire per ciascun nodo l'identificativo del rispettivo padre, nella topologia ad albero corrente.

I metadati degli eventi comprendono un nome, una firma e una stima della frequenza utilizzata durante il processo di ottimizzazione. I predicati definiti dall'utente presentano, oltre a nome e firma, una stima della loro selettività che è fornita dall'autore della funzione.

Di seguito sono elencati i metadati associati ad ogni attributo:

Metadati	Descrizione
Power	Costo per la lettura dell'attributo (in J)
Sample time	Tempo speso per la lettura dell'attributo (in s)
Constant	È un attributo con valore costante?
Rate of change	Velocità con cui varia il valore (units/s)
Range	Intervallo di valori acquisiti dall'attributo

Tali dati sono usati principalmente in due contesti:

- ottimizzazione delle query: Power, sample time e range
- disseminazione delle query ed elaborazione dei risultati: constant, rate to range

2.4.2 Tecnica 1: ordinamento delle letture e dei predicati

In questo paragrafo sarà descritto l'impiego dei metadati nell'ottimizzazione delle query.

Il campionamento di un sensore è spesso un'operazione costosa in termini di consumo di energia, ma allo stesso tempo indispensabile per poter valutare se un attributo, *sensors.s*, soddisfa o meno i predicati espressi nell'interrogazione. Se la verifica di un solo predicato è sufficiente a scartare una tupla della tabella, allora non sarà necessario

procedere con controlli relativi ad altri eventuali predicati, evitando quindi costi aggiuntivi. Tali verifiche, e in particolare l'azione di campionamento di cui necessitano, rappresentano quindi un costo non trascurabile e richiedono un accurato ordinamento.

Prima di procedere con la descrizione della tecnica di ottimizzazione adottata da TinyDB, si ricorda che:

- un attributo può essere valutato rispetto a più predicati
- tutti i predicati sono verificati sulla stessa tabella, *sensors*

Il campionamento di un sensore *t* viene trattato come un compito distinto, τ, da programmare insieme alla verifica dei predicati.
Si procede considerando un set di predicati $P = \{p_1, \ldots, p_m\}$ come sottoinsieme di un set operazioni $S = \{s_1, \ldots, s_n\}$, dove l'insieme differenza $S - P = \{\tau_1, \ldots, \tau_{n-m}\}$ contiene le operazioni di campionamento, una per ogni attributo referenziato in P.

Dato un predicato p, il suo fattore di selettività fp è dato dal rapporto tra il numero di tuple che soddisfano p e il numero di tuple della relazione cui p si riferisce.

La selettività delle operazioni di campionamento è sempre uguale a 1, mentre la selettività delle operazioni di selezione si ottiene assumendo che gli attributi abbiano una distribuzione uniforme all'interno del proprio range di valori.
Il costo di un'operazione può essere determinato consultando i metadati.

Definito l'insieme S e la selettività dei suoi elementi, si procede stabilendo un ordine parziale al suo interno: se p_i viene eseguito su un attributo campionato da τ_i, allora τ_i deve precedere cronologicamente l'esecuzione di p_i.

Oltre ai predicati nella clausola WHERE, le operazioni di campionamento devono essere ordinate anche rispetto alle clausole SELECT, GROUP BY e HAVING.
Come per i predicati di selezione, viene imposto un ordine parziale in modo tale che τ_i preceda ogni aggregazione, GROUP BY o HAVING che utilizza l'attributo *i*. Si noti

che le proiezioni non richiedono l'accesso al valore di *i*, per cui non devono essere incluse nell'ordinamento parziale.

L'ordine parziale completo di cui dovrà tenere conto l'ottimizzatore durante la pianificazione delle interrogazioni è il seguente:

1. Acquisizione dell'attributo *a*
2. Selezione
3. GROUP BY
4. aggregazione
5. HAVING

Naturalmente anche questi operatori devono essere inseriti nell'insieme delle operazioni S con gli appropriati costi e selettività.

Exemplary aggregate pushdown

Consideriamo la query:

SELECT WINMAX(light, 8s, 8s)
FROM *sensors*
WHERE mag > *x*
SAMPLE PERIOD 1s

Ogni 8 secondi viene riportata la luminosità massima rilevata negli ultimi 8s, ma è accettata solo quella per cui il corrispondente valore di *mag* è superiore al valore *x*.
È interessante notare che, a meno che il predicato "*mag* > *x*" non sia molto selettivo, sarà meno costoso valutare la query col seguente approccio:
Si confronta ogni nuovo valore dell'attributo *light* con l'ultimo campionato. Se il valore di luminosità è inferiore, la tupla viene scartata; se la luminosità è maggiore, allora si può campionare anche il valore di *mag* ed effettuare il controllo sul valore letto.

Questo procedimento può essere applicato ogni qualvolta ci si trovi in presenza di *exemplary aggregates* (e.g. MIN, MAX), ma anche nel caso di *non-windowed* aggregates, durante l'esecuzione di aggregazioni all'interno della rete.
Si suppone, ad esempio, di dover eseguire un'aggregazione in un nodo intermedio nell'albero di routing. Se la query contiene un predicato la cui valutazione richiede un'acquisizione abbastanza onerosa in termini di energia, allora, prima di procedere con tale operazione, è più conveniente valutare se il valore locale potrebbe influire sul valore dell'aggregazione.
Per semplicità, assumiamo che gli attributi coinvolti nell'aggregazione (e.g. light) siano campionati dalla stessa distribuzione. Per cui, per aggregazioni di tipo MIN e MAX, la probabilità che il secondo di due campionamenti fornisca un valore inferiore al primo è pari a 0.5. Per *n* campionamenti, la probabilità che l'ennesimo rappresenti il valore riportato dalla funzione è quindi pari a $1/0.5^{n-1}$.
Per poter calcolare i benefici ottenibili dall'applicazione di un *exemplary aggregate pushdown,* definiamo i seguenti parametri, i cui valori sono disponibili nel catalogo dei metadati presente in ciascun nodo:
- **a** Attributo su cui opera l'aggregazione
- **S(a)** Stima della selettività fatta dall'aggregazione su **a**
- **C(a)** Costo di acquisizione di **a**
- **P** Selettività di aggregazione dei predicati della query su un set di attributi p_i
- **K** costo per l'acquisizione degli attributi p_i

SELECT MAX(a)
FROM *sensors*
WHERE $p_1, p_2, \ldots, p_n$
SAMPLE PERIOD 1s

Valutiamo ora i consumi nei due casi, prima senza, poi con pushdown:

- senza pushdown: K + P * C(a) (1)
- con pushdown: C(a) + S(a) * K (2)

Quando (2) è minore di (1), è vantaggioso applicare la tecnica del pushdown

2.4.3 *Tecnica 2:* Stream di eventi per la conservazione dell'energia

Come secondo esempio, consideriamo l'ottimizzazione della query seguente:

ON EVENT *e* (nodeid)

SELECT a_1

FROM *sensors* AS s

WHERE s.nodeid = *e*.nodeid

SAMPLE PERIOD *d* FOR *k*

Ogni occorrenza dell'evento *e* provocherà l'avvio di una nuova istanza della query interna. Tale istanza campiona i dati ogni *d* secondi, per una durata complessiva di *k* secondi, poi termina. Si noti che questa istruzione potrebbe portare alla presenza simultanea di tante istanze attive della stessa query. Se il numero dovesse diventare elevato, ne conseguirebbe che il beneficio introdotto da tali interrogazioni (*query based on events*), ossia l'assenza di polling, sarebbe annullato dall'ingente consumo di energia da parte delle diverse istanze durante il campionamento e la trasmissione dei risultati.

Per ovviare a questo problema, TinyDB offre una tecnica di ottimizzazione *multi-query* che consiste nella conversione degli eventi esterni di tipo *e* in un flusso di eventi e nella riscrittura dell'intero set di query interne in un *sliding window join,* tra *events* e *sensors,* con una dimensione di *k* secondi sullo *stream* degli eventi.

Ad esempio:

SELECT s.a1

FROM *sensors* AS s, events AS e

WHERE s.nodeid=e.nodeid

AND e.type= *e*

AND s.time – e.time <= *k* AND s.time > e.time

SAMPLE PERIOD *d*

La query dell'esempio precedente diventa quindi un join tra *sensors* e un punto di materializzazione di dimensione k su *events*. Al momento del suo arrivo, ogni nuova tupla *events* viene aggiunta al buffer degli eventi. Quando viene creato un nuovo record

s in *sensors*, gli eventi più vecchi di *k* secondi sono eliminati dal buffer e *s* è posta in join con gli eventi restanti.

Il vantaggio di questo approccio consiste nell'avere una sola query eseguita per volta.

La figura mostra il consumo di potenza per interrogazioni basate sugli eventi, nel caso asincrono originario e in quello rielaborato basato sullo *stream* di eventi.

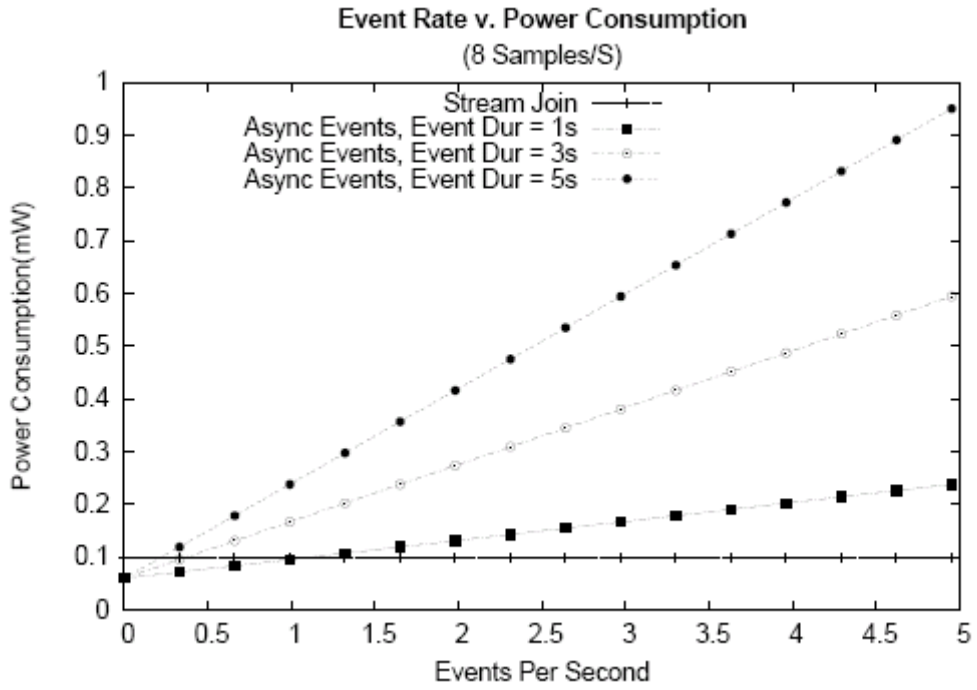


Figura 7: consumo per l'esecuzione di query basate sugli eventi: eventi asincroni vs. join

Osservando la figura, possiamo notare che, a seconda della frequenza con cui si verificano gli eventi, uno dei due approcci tende ad essere preferibile. In particolare, per occorrenze frequenti (almeno una per secondo) il nuovo metodo è chiaramente migliore in termini di consumo energetico.

Le applicazioni per il monitoraggio delle vibrazioni possono essere un esempio di questo comportamento, in quanto viene segnalato un evento ogni qualvolta che un segnale supera o scende sotto una certa soglia.

L'approccio asincrono è invece preferibile nel caso in cui l'occorrenza degli eventi sia molto bassa (meno di un evento per secondo); in tale circostanza infatti, nel caso rielaborato del flusso di eventi, si assisterebbe ad un continuo e spesso inutile controllo del buffer per verificare la presenza di un eventuale evento. Tuttavia, basta un piccolo

incremento del tasso di occorrenza per aumentare considerevolmente il consumo di energia.

Il nuovo metodo introduce un'ulteriore variazione rispetto a quello originario.
Nella formulazione iniziale infatti, l'inizio del campionamento per le query interne ha come base l'occorrenza dell'evento: se, ad esempio, l'evento e_1 si presenta all'istante t, le letture saranno effettuate agli istanti $t + d$, $t + 2d$, ...; se un secondo evento e_2 si presenta all'istante $t + i$, produrrà un diverso insieme di letture agli istanti $t+i+d$, $t+i+2d$, ... Questo comporta, a meno che i non sia uguale a d (e.g. eventi in fase), uno sfasamento dell'esecuzione delle due query di un tempo massimo di d secondi. Nel nuovo approccio invece, lo *stream* di tuple appartenenti alla tabella *sensors* è condiviso da tutti gli eventi.
Nella maggior parte dei casi l'utente non deve preoccuparsi del fatto che le tuple non siano in fase con gli eventi, ma in alcune circostanze può essere molto importante. Il sistema può ovviare a questo problema aumentando la frequenza di campionamento (*oversampling*) o effettuando campionamenti aggiuntivi ogni d secondi. Questo comporta un aumento dei costi di acquisizione, che tuttavia potrebbe essere ancora inferiore rispetto all'esecuzione simultanea di più query.
Al momento, per semplicità, l'utente può imporre l'allineamento, specificando la clausola ON ALIGNED EVENT.

2.5 Disseminazione delle interrogazioni

Una volta prodotto il piano di esecuzione ottimale della query, l'interrogazione è distribuita, con modalità *broadcast*[2] (diffusa) , dalla radice a tutti i nodi della rete. Non appena un nodo riceve la query, deve decidere se applicarla localmente e/o inoltrarla ai nodi figli oppure scartarla.
Diremo che una query q è applicabile a un nodo n se esiste una probabilità non nulla che n produca un risultato per q.

[2] Per **broadcasting** si intende la trasmissione di informazioni da un sistema trasmittente ad un insieme di sistemi riceventi non definito a priori. Nel caso in questione, la query trasmessa dal nodo viene avvertita da tutti i dispositivi che si trovano nelle vicinanze (a distanza di un hop dal mittente).

Un'errata valutazione dell'esecuzione delle query in ambienti distribuiti come TinyDB può causare ingenti sprechi di energia.
Se un'interrogazione non deve essere applicata in un particolare nodo e tale nodo non ha figli sui quali poter essere eseguita, allora l'intero sotto-albero, avente come radice il nodo stesso, sarà escluso dal campo di applicazione della query, risparmiando così i costi di distribuzione, esecuzione ed inoltro dei risultati attraverso diversi nodi.
Si pone quindi il problema di dover stabilire quando un dispositivo e i suoi nodi figli devono essere esclusi dal campo di applicazione della query.
Se un nodo è a conoscenza del fatto che nessuno dei suoi figli può al momento soddisfare la query ricevuta, allora non deve inoltrare ulteriormente l'interrogazione. In questa sezione sarà proposta una struttura dati, denominata *Semantic Routing Tree* (SRT), che consente a ciascun dispositivo di tenere traccia degli attributi, costanti o variabili, dei dispositivi sottostanti nell'albero di routing.

2.5.1 Semantic Routing Tree

Un *Semantic Routing Tree* (SRT) è una struttura ad albero realizzata per consentire a ciascun nodo di determinare efficacemente se i dispositivi figli possono o meno essere interessati dall'esecuzione della query data su un attributo costante **A**. Tradizionalmente la costruzione dell'albero di routing nelle reti di sensori avviene nel seguente modo: ciascun nodo sceglie come padre quello che assicura un miglior collegamento verso la radice. Negli SRT tale scelta deve includere anche alcune considerazioni di tipo semantico. In generale questa struttura è meglio applicabile quando, per un nodo, ci possono essere diversi dispositivi padre con qualità di collegamento simile.
Concettualmente un SRT è un indice sull'attributo **A** che può essere utilizzato per localizzare i nodi in possesso di dati rilevanti per la query. Ogni dispositivo memorizza un singolo intervallo unidimensionale rappresentate il range complessivo dei valori assunti da A nei nodi a valle.
Quando una query *q* con un predicato su **A** arriva al nodo *n*, il dispositivo controlla se alcuni dei figli assumono valori di **A** che si sovrappongono al range espresso nel predicato della query.

Se il controllo da esito positivo, *n* si prepara a ricevere i risultati e a ritrasmettere la query, mentre, in caso contrario, non procede con la ritrasmissione. In entrambi i casi, se la query è applicabile localmente, *n* stesso procede con la sua esecuzione e il risultato finale sarà trasmesso ai nodi a monte insieme a quelli eventualmente forniti dai dispositivi sottostanti. Nel caso in cui né il nodo in questione, né i nodi figli possano applicare l'interrogazione, allora quest'ultima sarà semplicemente ignorata.

La costruzione di un SRT comprende due fasi:

1) la prima fase consiste nella diffusione dell'*SRT build request* (richiesta di costruzione dell'SRT) a partire dalla radice e via via ritrasmessa sino a quando tutti i nodi della rete non l'avranno ricevuta. Questa richiesta comprende il nome dell'attributo **A** in base al quale deve essere definito l'albero. Dal momento che la richiesta viene diffusa in modalità *broadcast* dalla radice verso i nodi periferici della rete, è possibile che per un nodo si presentino diverse possibili alternative di padre.
2) Captata la richiesta, se un nodo *n* ha figli, ritrasmette loro il messaggio e resta in attesa delle loro eventuali risposte per un tempo determinato. Tali risposte, i *parent selection message,* contengono il valore dell'attributo **A** assunto dal figlio e il suo identificativo. Per ogni risposta ricevuta, il nodo registra le informazioni ottenute dai figli. Terminata l'acquisizione dei valori da parte di tutti i nodi sottostanti, *n* sceglie a sua volta un padre e gli invia un parent selection message indicante l'intervallo di valori di A coperto dal nodo stesso e dai suoi discendenti. Il nodo padre registra il range di valori ottenuto, l'identificativo del mittente, quindi procede, a sua volta, con la scelta di un padre a cui inviare i valori. Queste operazioni sono ripetute sino all'arrivo dei dati alla radice della rete, sino a quando cioè l'SRT non sarà completamente definito.

   Se, al contrario, il nodo che riceve un SRT built message non ha discendenti, sceglie direttamente un padre *p* tra quelli disponibili e procede inviandogli il proprio valore dell'attributo A, inserito all'interno di un parent selection message.

Dal momento che i figli potrebbero smettere di funzionare o essere stati rimossi, i nodi sono dotati di un timeout che indica il massimo tempo d'attesa tollerato dal padre per ricevere una risposta dai nodi a valle. Trascorso questo periodo, i figli che non forniscono una risposta sono eliminati dalla lista, *child list*. Se, tuttavia, il figlio riesce comunque a trasmettere i propri dati, sebbene fuori tempo massimo, esso viene incorporato nell' SRT come se fosse un nuovo nodo.

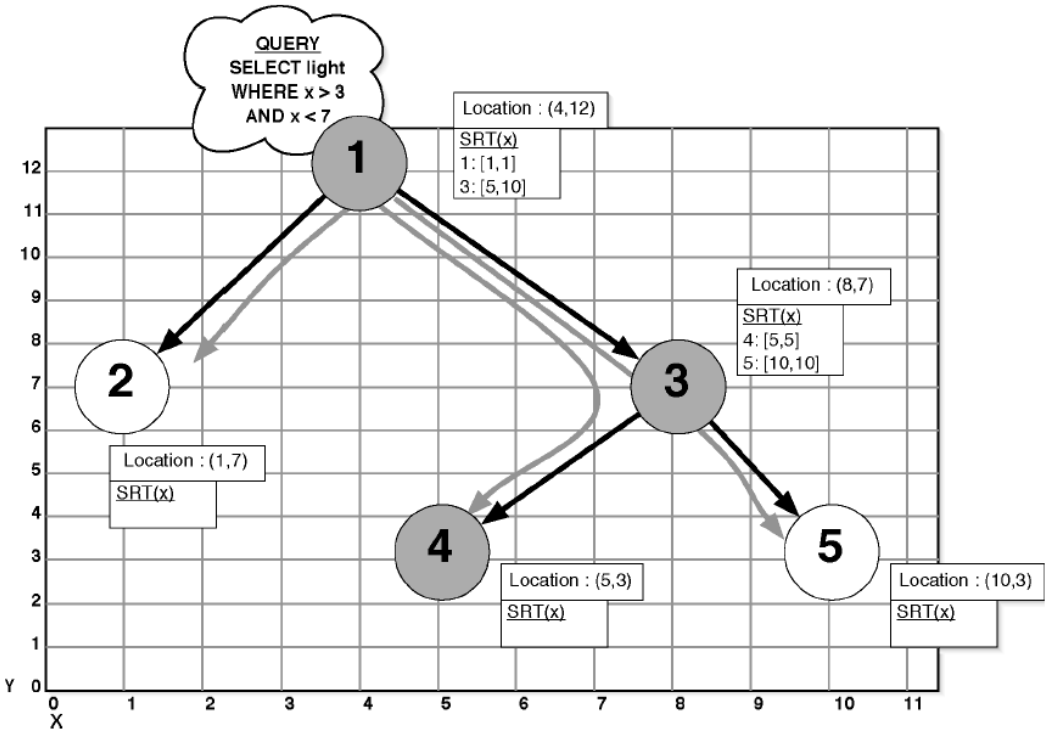


Figura 8: Esempio di uso di un SRT in una Query

La figura mostra un SRT costruito sulla coordinata X, appartenente ad ogni nodo posto in un piano cartesiano. La query, arrivata alla radice, viene propagata in tutta la rete. I nodi grigi sono gli unici ad entrare nel campo di applicazione della query. Si noti che il nodo 3, pur non presentando un valore di X valido per l'interrogazione, è necessario per poter raccogliere il dato valido del nodo 4.

L'SRT svolge un ruolo analogo a quello svolto dagli indici nei database tradizionali. La sintassi per la sua creazione è la seguente:

CREATE SRT loc ON *sensors* (xloc, yloc) *ROOT* 0

"*ROOT* 0" indica l'identificativo del nodo a partire dal quale deve essere creato l'SRT.

2.5.2 Gestione dell'SRT

Le operazioni di gestione dell'SRT riguardano principalmente le tre seguenti situazioni:

- inserimento di nuovi nodi
- variazioni nella qualità dei collegamenti
- scomparsa di nodi esistenti

L'inserimento di un nuovo nodo o variazioni nella qualità dei collegamenti potrebbero indurre un nodo alla scelta un nuovo padre.
Per fare ciò, il nodo invia un *parent selection message* al nuovo genitore *n.* Se questa operazione comporta una variazione dell'intervallo di valori registrato da *n,* a sua volta il nodo notifica l'avvenuta modifica al rispettivo padre. In questo modo, l'aggiornamento dell'SRT viene propagato sino alla radice dell'albero.

Per gestire la scomparsa di un nodo figlio dall'SRT, i nodi padre associano ad ogni discendente due parametri: *active* query *id* e *last epoch.*
Quando un padre *p* inoltra una query *q* a un figlio *c*, imposta il suo valore della query attiva pari all'identificatore di *q* e imposta il campo *last epoch* a 0. Ogni volta che *p* inoltra o aggrega i risultati da *c*, aggiorna il valore di tale campo con l'istante in cui è stato ricevuto il risultato.
Se, per un certo numero di periodi (*epoch*) *t,* il padre *p* non riceve segnali dal figlio *c*, può presupporre che per qualche motivo *c* non sia più presente, quindi elimina i dati da questo ricevuti. Successivamente invia ai figli restanti una richiesta di ritrasmissione dei diversi intervalli di validità, quindi procede ricalcolando il range di valori per l'attributo valutato nella query. Se questo nuovo intervallo è diverso da quello precedente, è necessario che questo cambiamento sia notificato al rispettivo padre e via via fino alla radice.

Grazie alla regole di gestione appena descritte, sarebbe possibile applicare l'SRT anche ad attributi non costanti. Tuttavia, se i loro valori mutano con una certa frequenza, i costi di aggiornamento della rete potrebbero diventare proibitivi.

2.5.3 Valutazione dei benefici dell'SRT

I benefici apportati da un SRT dipendono principalmente dalle politiche di clustering (raggruppamento) adottate, ossia dalla qualità dei raggruppamenti effettuati a valle dei nodi padre.
Di seguito sono presentate tre diverse politiche con cui i dispositivi, durante la fase di definizione dell'SRT, possono scegliere il rispettivo padre tra quelli disponibili.

Primo approccio – *random* (casuale) ogni nodo sceglie casualmente un padre tra i dispositivi vero i quali il collegamento è più sicuro.

Secondo approccio – *closest parent* (padre più vicino) ogni padre riporta il valore del proprio attributo indice nell'*SRT-built request* e ogni figlio sceglie come padre quello che presenta un valore più vicino al proprio.

Terzo approccio – *clustered*: la scelta del padre avviene in modo analogo al closest approach, con la seguente variazione: se un nodo si accorge che un altro dispositivo (fratello) sta selezionando un padre tramite l'invio di un *parent selection message*, "spia" il messaggio per controllare la scelta effettuata e il valore dell'attributo inviato. Il nodo sceglie quindi il proprio padre (che può essere lo stesso di uno dei suoi fratelli) in modo tale da minimizzare la diffusione dell'intervallo dei valori acquisiti dall'attributo al di sotto di uno stesso nodo.

I ricercatori della Berkeley hanno sperimentato le tre politiche in tre tipi di distribuzione:

- Distribuzione casuale: ogni valore dell'attributo è scelto casualmente e uniformemente all'interno dell'intervallo [0, 1000].
- Distribuzione geografica: i valori (unidimensionali) assunti dai sensori sono calcolati in base alle coordinate cartesiane (x, y) dei nodi posti all'interno di una griglia. In questo modo, il valore di un sensore tende ad essere altamente correlato ai valori dei nodi vicini.

- Distribuzione reale: per l'esperimento è stata utilizzata una topologia di rete basata sui dati acquisiti da una rete di 54 mote distribuiti presso l'Intel-Research, Berkeley Lab.

I risultati sono valutati in termini di nodi attivi; i nodi inattivi non comportano un costo per la query data, ma solo un piccolo consumo di energia che consente al loro processore di restare in uno stato di *idle* (inattivo), in attesa di nuove interrogazioni.

Le tre politiche di raggruppamento vengono messe a confronto con il caso migliore, quello cioè in cui si attivano solo i nodi i cui valori si sovrappongono al predicato della query, e con il caso senza SRT, nel quale tutti i nodi partecipano all'esecuzione di ogni query.

I risultati sperimentali ottenuti dimostrano che nelle tre distribuzioni studiate (casuale, geografica e reale) il clustered approach risulta sempre quello che più si avvicina al caso migliore. In particolare nella topologia reale e in quella geografica, tale approccio è molto prossimo al caso ottimo.

2.5.4 Costi di gestione dell'SRT

I costi di mantenimento dell'SRT comprendono i costi di costruzione dell'albero e i costi di gestione.

I costi di costruzione sono paragonabili a quelli nelle reti di sensori tradizionali (che presentano un albero di routing), anche se leggermente superiori per il fatto che i messaggi per la scelta del padre sono inviati esplicitamente, mentre nella altre reti di sensori i nodi padre non richiedono sempre una conferma dai figli.

Per quanto riguarda i costi di gestione, la figura seguente mostra i risultati ottenuti dai ricercatori della Berkeley.

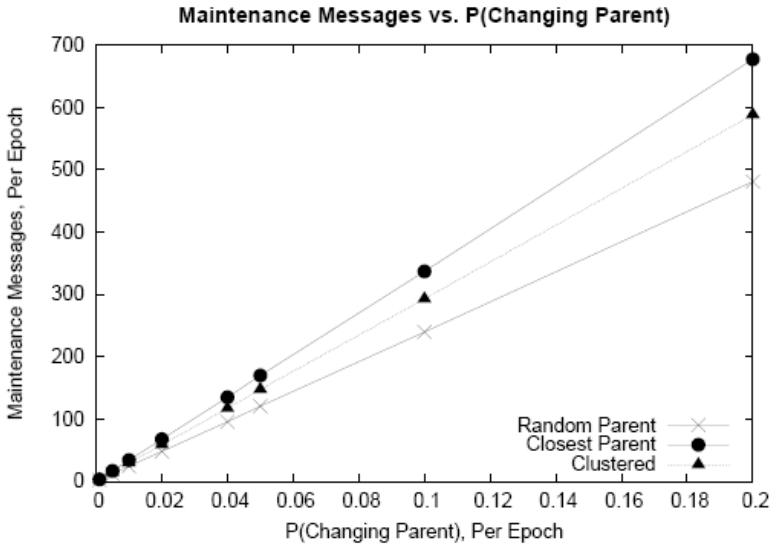


Figura 9: costi di gestione per diverse politiche di scelta del padre

L'asse x rappresenta l'ammontare dei movimenti (scelta di un nuovo padre) che avvengono nella rete.
L'asse y rappresenta invece l'ammontare dei messaggi inviati nella rete come conseguenza di questi movimenti.

Notiamo che se il tasso di movimento (scelta di un altro padre) rimane sufficientemente basso, il conseguente numero di messaggi nella rete, e quindi i costi di mantenimento dell'approccio SRT, saranno abbastanza limitati da rendere questa tecnica ancora attraente.
Un aumento dei movimenti nella rete porta però a un rapido aumento dei costi, rendendo meno chiari i benefici introdotti dall'SRT.

La figura mostra inoltre come le tre diverse politiche adottate per la costruzione dell'SRT portino a diversi costi di gestione. Questo si verifica perché, a seconda dell'approccio scelto, variano la profondità media dell'albero (7.67 per quello Random, 10.47 per il Closest e 9.2 per il Clustered) e la diffusione dei valori in un particolare sotto-albero. Una profondità maggiore, generalmente, porta ad avere un maggior numero di messaggi che risalgono l'albero.

Il closest approach rappresenta da questo punto di vista il caso peggiore, in quanto non viene data nessuna preferenza nella scelta del padre, tra quelli disponibili con un ampio range di valori.

2.5.5 Osservazioni

Gli SRTs forniscono un meccanismo efficiente per la distribuzione delle query nella rete e per il recupero dei risultati di interrogazioni eseguite su attributi costanti.
Per i valori che presentano un alto grado di correlazione in nodi vicini, SRT può ridurre, di quasi un ordine di grandezza, il numero di dispositivi che devono distribuire le query e inoltrare il continuo flusso di risultati ottenuti dai nodi figli.
Il vantaggio dell'SRT consiste nel non dover raccogliere alla radice tutte le informazioni riguardanti i valori dei sensori e la topologia della rete; tale operazione sarebbe infatti abbastanza costosa e risulterebbe difficile mantenere la consistenza di dati quali il mutamento dei valori acquisiti e la connettività.
I costi di mantenimento sono ragionevoli per almeno alcune applicazioni nel mondo reale, ma a differenza dei routing tree tradizionali, è necessario tenere conto del tasso di *parent-switching* (cambio di padre).

2.6 Elaborazione delle interrogazioni

Terminate le fasi di disseminazione e ottimizzazione delle interrogazioni, il query *processor* inizia la fase di esecuzione.
Il primo paragrafo descriverà semplicemente l'esecuzione delle interrogazioni, mentre, nei paragrafi successivi, si tratteranno i problemi della definizione di un ordine di priorità nella trasmissione dei risultati e dell'adattamento dei tassi trasmissivi e di campionamento alle condizioni della rete e alla potenza disponibile.

2.6.1 Esecuzione delle interrogazioni

L'esecuzione delle query consiste in una semplice sequenza di operazioni eseguite in ogni nodo durante ogni periodo (epoch).

1) I nodi restano in uno stato di *sleep* (sonno) per più di un *epoch*
2) Si svegliano
3) Campionano i sensori
4) Applicano gli operatori ai dati prodotti localmente e a quelli acquisiti da altri nodi
5) Inoltrano i risultati ai rispettivi padri

I nodi restano nello stato di *sleep* il più a lungo possibile per minimizzare i consumi e si svegliano solo per campionare i sensori e per ritrasmettere e distribuire i risultati.
Essendo i dispositivi sincronizzati temporalmente, un nodo padre può garantire di essere sveglio quando un figlio tenta di inviare un messaggio. L'arco di tempo, t_{awake}, durante il quale un nodo deve essere sveglio, in modo da poter portare a termine i compiti precedentemente illustrati, dipende principalmente dal numero di nodi che trasmettono contemporaneamente nello stesso canale radio.
TinyDB utilizza un semplice algoritmo per ridurre t_{awake}, basato sulla dimensione del "vicinato", ottenuta misurando il traffico nei nodi vicini. Tuttavia, ci sono situazioni nelle quali un nodo potrebbe essere costretto a combinare o ad eliminare dei risultati come conseguenza di un intervallo di campionamento o di un t_{awake} troppo brevi per consentirgli di completare le operazioni di calcolo e trasmissione necessarie.

Una volta sveglio, il nodo inizia a campionare i sensori e a filtrare i risultati in accordo col piano fornito dall'ottimizzatore. Le letture sono eseguite con un tasso di campionamento appropriato per la query, basato sul calcolo del tempo di vita, sulle informazioni relative all'occupazione del canale radio e al consumo di energia.

Pianificazione delle comunicazioni e Interrogazioni di aggregazione

Quando si eseguono query aggregate bisogna porre particolare attenzione nel coordinare i periodi in cui padri e figli sono svegli, in modo tale che i primi possano avere accesso alle letture dei discendenti prima che queste siano aggregate.
L'idea di base consiste nel dividere un epoca in periodi e di assegnare ogni nodo a uno specifico periodo in base alla propria posizione all'interno dell'albero.

TinyDB utilizza questo approccio per tutti i tipi di query, grazie all'efficiente gestione del canale radio e alle buone caratteristiche in termini di consumi.
Gli intervalli hanno tutti la stessa durata e sono numerati in ordine inverso (l'intervallo 1 è l'ultimo dell'epoca).
Ciascun nodo è assegnato all'intervallo con numero uguale al proprio livello e, durante il periodo precedente al proprio, i dispositivi restano in ascolto sui rispettivi canali, acquisendo i risultati dai nodi figli.
Quando giunge l'intervallo assegnato, in caso di aggregazione, il nodo calcola il *partial state record* (record di stato parziale) che consiste nella combinazione dei risultati ottenuti dai figli con le letture eseguite localmente.
Eseguito il calcolo, ritrasmette nella rete il record di stato parziale o le semplici letture eseguite dai propri sensori. In questo modo, l'informazione risale la rete sino a raggiungere la radice durante l'intervallo 1.

Esempio:
La figura rappresenta lo schema di una query che riporta il numero di nodi nella rete.
Nella parte di destra è rappresentata la struttura delle rete con la classificazione dei nodi ai diversi livelli.
Nella parte di sinistra sono invece rappresentati gli intervalli nei quali è stata suddivisa l'epoca e i nodi ad essi associati. I dispositivi trasmettono (operazione indicata da una freccia) durante l'intervallo corrispondente al proprio livello. I numeri nel cerchio sulla freccia rappresentano il conteggio contenuto in ogni record di stato parziale.

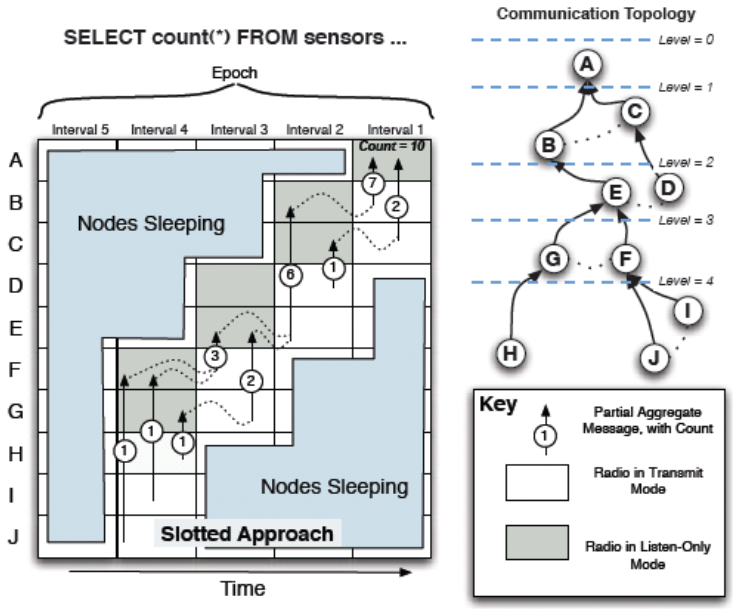


Figura10: record di stato parziale che risalgono l'albero di routnig nel corso di un epoch

2.6.2 Priorità nella consegna dei dati

I dati, una volta campionati e rielaborati localmente, vengono inseriti in una coda all'interno del canale radio, in attesa di essere recapitati al nodo padre. All'interno del buffer sono inserite anche le tuple inoltrate da tutti gli altri nodi della rete. Se il traffico è ridotto e le frequenze dei dati sono sufficientemente basse, la coda non va incontro a problemi di intasamento e tutti i nuovi risultati trasmessi dai nodi riescono a trovare spazio al suo interno.

Tuttavia, data la variabilità dei messaggi prodotti all'interno della rete, il buffer può andare in uno stato di *overflow* (eccedenza di dati) e, in tale situazione, il sistema deve decidere se scartare delle tuple o combinarle tra loro.

La capacità di poter stabilire a tempo di esecuzione il valore di un dato è di centrale importanza in sistemi ACQP (*ACquisitional* Query *Processing*) per gli alti costi di acquisizione e trasmissione e a causa di queste situazioni, nelle quali il tasso con cui i dati arrivano a un nodo supera quello massimo di trasmissione dei risultati.

Un semplice approccio concettuale adottato da TinyDB è il seguente:

ogni volta che il sistema è pronto ad inoltrare un tupla, invia il risultato che perfezionerà maggiormente la risposta attesa dall'utente. Chiaramente la metrica di valutazione

dipende dall'applicazione: per una query di aggregazione, ad esempio, si può scegliere di minimizzare l'intervallo di confidenza dei valori per ciascun raggruppamento.

Politiche per interrogazioni selettive

Consideriamo inizialmente tre semplici schemi di priorità utilizzabili in normali query di selezione:

Naive: nessuna tupla ha priorità sulle altre, per cui i dati escono dalla coda, secondo una tecnica FIFO (First-In First-Out). In caso di overflow, le tuple in arrivo che trovano il buffer pieno sono eliminate.

Winavg: questo schema è simile al precedente, eccetto per il fatto che, in caso di overflow, le due tuple in testa alla coda sono fuse in una unica, calcolandone la media, in modo tale da consentire a nuovi dati l'accesso al buffer.

Delta: ad ogni tupla è assegnato un punteggio iniziale calcolato sulla base della sua differenza dall'ultimo record trasmesso con successo dal nodo e, ad ogni istante di tempo, viene consegnata la tupla con punteggio maggiore. Quando il buffer si trova in condizioni di overflow, viene scartato il dato con punteggio inferiore.
Questo schema si basa sul fatto che probabilmente i cambiamenti maggiori sono più interessanti.

Politiche per interrogazioni di aggregazione

Di seguito saranno illustrate una serie di tecniche basate sullo *snooping* (curiosare) , implementate sul sistema TinyDB per gestire le priorità dei dati nelle query aggregate.

Si considera ad esempio l'interrogazione

SELECT f_{agg} (a_1)
FROM *sensors*
GROUP BY a_2
SAMPLE PERIOD x

che calcola f_{agg} applicata al valore di a_1, prodotto da ogni nodo ogni x secondi.

Per interrogazioni con pochi gruppi, si può applicare la seguente tecnica:

Snooping: questo approccio consente ai nodi di sopprimere i valori aggregati localmente, sfruttando la semantica delle funzioni di aggregazione e i risultati trasmessi dai nodi vicini sul canale radio. Questa tecnica è resa possibile dalla natura *broadcast*[2] del canale.
Consideriamo ad esempio una query con funzione di aggregazione MAX applicata a un attributo *a*.
Se un nodo *n*, in ascolto sul canale, percepisce un valore di *a* maggiore di quello da lui ottenuto applicando la funzione MAX localmente, allora assegna un punteggio basso al proprio record oppure lo elimina direttamente, dal momento che avrebbe scarsa probabilità di essere un massimo.
Al contrario, se *n* si rende conto del fatto che il risultato ottenuto, eseguendo localmente la funzione MAX, è maggiore di molti altri valori parziali percepiti nella rete, allora assegna al proprio record un punteggio alto, poiché in questo caso ha invece buona probabilità di essere un massimo.

La seguente figura mostra un esempio di questo tipo. Il nodo 2 può assegnare un punteggio basso al proprio record dopo aver ascoltato il valore di MAX calcolato al nodo 3, dal momento che risulta essere maggiore del proprio.

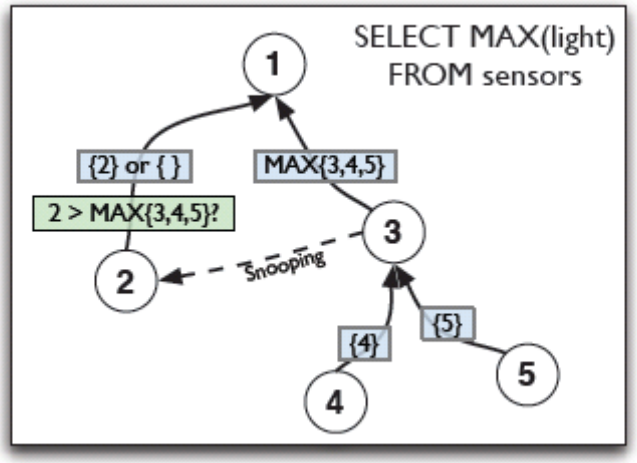


Figura 11: lo *snooping* riduce i dati che i nodi devono spedire in query di aggregazione

Questa politica è applicabile a tutte le aggregazioni monotone (MIN, MAX, TOP-N, etc.) poiché è possibile stabilire deterministicamente se il valore calcolato localmente in un nodo ha la possibilità di apparire nella risposta finale.
In reti con un alta densità di nodi, questa tecnica consente di ridurre drasticamente le comunicazioni.

È inoltre possibile impiegare questa politica anche per altre tipologie di aggregazione. Ad esempio, in una query con la funzione AVERAGE, un nodo può decidere di assegnare un punteggio basso al risultato elaborato localmente se, ascoltando i messaggi inviati sul canale, percepisce che molti nodi "fratelli" hanno ottenuto risultati simili al suo. Per questo approccio di lavoro, i nodi padre devono memorizzare il numero dei figli da cui hanno ricevuto messaggi recentemente.
Assumono quindi che la mancata ricezione del valore della media da parte di alcuni figli sia dovuta al fatto che il valore da loro calcolato è simile a quello dei fratelli.

2.6.3 Adattamento del tasso di trasmissione e del consumo energetico

Nel paragrafo precedente abbiamo visto come TinyDB riesca a sfruttare la semantica della query per trasmettere i dati più rilevanti nel caso in cui l'energia o la larghezza di banda siano limitate.
In questo paragrafo saranno illustrate le tecniche, adottate da tinyDB, per la scelta e l'adattamento dei tassi di lettura e trasmissione, al fine di limitare la perdita di pacchetti nella rete e l'overflow delle code. Grazie alla capacità del sistema di regolare questi tassi, è possibile ridurre notevolmente la frequenza con cui devono essere prese le decisioni sulla priorità dei dati.
Durante la prima fase di ottimizzazione dell'interrogazione, il query processor sceglie il tasso di campionamento e quello di trasmissione basandosi sulle condizioni di carico correnti della rete e sulle richieste relative al tempo di vita e alla frequenza di lettura dei dati. Naturalmente queste decisioni, prese staticamente all'inizio dell'elaborazione della query, non possono più considerarsi valide dopo un certo periodo di funzionamento.

TinyDB deve essere quindi pronto a reagire a cambiamenti nelle condizioni. D'altra parte, un mancato adattamento potrebbe portare alla paralisi del sistema, riducendo drasticamente il flusso dei dati o causando gravi sprechi di energia.

Il sistema può effettuare due tipologie di monitoraggio: adattamento al traffico della rete e al consumo di potenza.

Traffico nella rete

TinyDB monitorizza il canale e, non appena il traffico aumenta, provvede a ridurre il numero di pacchetti trasmessi.
Dalla figura è possibile notare come il numero complessivo dei pacchetti distribuiti da quattro nodi, che trasmettono contemporaneamente con un alto tasso, sia sostanzialmente inferiore al caso in cui gli stessi nodi trasmettono ad un tasso inferiore.
Notiamo anche che i *4* mote *adaptive* non presentano la stessa caduta di prestazioni, in quanto il sistema monitorizza il canale e adatta il tasso trasmissivo alla sua occupazione.

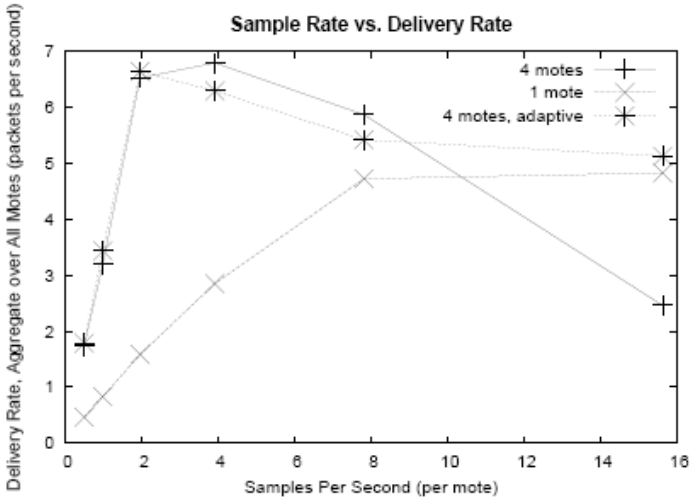


Figura 12: tasso di campionamento vs. tasso di trasmissione

La riduzione del tasso di trasmissione porta rapidamente la coda in condizioni di saturazione, forzando TinyDB ad eliminare o ad aggregare tuple. D'altra parte, se il

sistema non riducesse tale tasso, sarebbero consegnati complessivamente meno pacchetti. In aggiunta, la capacità di scegliere quali tuple eliminare sfruttando la sintassi delle interrogazioni rende TinyDB in grado di migliorare considerevolmente la qualità dei risultati forniti all'utente finale.

Misura del consumo energetico

Ci si concentra ora sul problema di dover regolare il tasso di trasmissione delle tuple anche in rapporto ai requisiti sul tempo di vita, in risposta a un incorretto calcolo del tasso di campionamento, eseguito durante l'ottimizzazione iniziale della query.

Dal momento che è possibile calcolare una previsione di quella che sarà la tensione della batteria a un istante t durante l'esecuzione della query, il sistema può confrontare il suo voltaggio corrente con quello calcolato in precedenza.
Assumendo che la tensione decada linearmente, è possibile rivalutare la caratteristica del consumo di potenza del dispositivo e successivamente ricalcolare il tempo di vita.

Conclusioni e possibili sviluppi

Abbiamo visto che le reti di sensori rappresentano una tecnologia ancora giovane, ma che, grazie al progresso compiuto nel campo della miniaturizzazione degli apparati di elaborazione, memorizzazione e comunicazione, inizia a riscuotere un interesse sempre maggiore in vari settori, da quello militare a quello di monitoraggio ambientale, da quello industriale a quello agricolo, sino ad offrire un valido supporto per la medicina, ma anche per la vita domestica.
Abbiamo visto che un sistema come TinyDB, in grado di fornire uno strato di isolamento tra le applicazioni e le specifiche implementazioni delle risorse o dei servizi al di sotto di esse, si presenta come un supporto quasi indispensabile "per quegli scienziati o ingegneri civili ai quali non si può chiedere di programmare un mote e di progettare una rete" (Intel Research scientist Wei Hong).
TinyDB crea infatti un'interfaccia tra utente e dispositivi, che consente all'utilizzatore di analizzare i dati raccolti dai sensori come se stesse interrogando un database tradizionale.
Senza questo strumento, l'amministratore della rete dovrebbe scrivere centinaia di linee di codice per raccogliere i dati da ogni mote, coordinare il modo in cui tali dati "rimbalzano" nella rete, aggregarli e inoltrarli a un PC. Una volta scritto, il codice dovrebbe poi essere installato su ogni nodo della rete.

Con TinyDB dettagli quali la gestione della rete e dei consumi, la sincronizzazione dei nodi e infine le decisioni riguardanti l'elaborazione dei dati, sono completamente nascoste all'utente, consentendo quindi un più facile dispiegamento della rete

Sono tuttavia trascorsi alcuni anni dallo sviluppo di *tiny database*, quali TinyDB e Cougar; da allora, l'idea di un linguaggio dichiarativo di alto livello per reti di sensori è stato ridefinito in vari modi.

Alcuni di questi sviluppi riguardano i seguenti campi:

- *Programmazione di rete dichiarativa*

  Tra i motivi che hanno portato al successo di TinyDB c'è sicuramente l'attrazione nei confronti di un linguaggio dichiarativo come SQL che nasconde all'utente i dettagli della rete e dei sensori. In molti casi questo può semplificare la programmazione, sostituendo centinaia di linee di codice in linguaggio C con poche linee di SQL. Tuttavia l'ambizione di questo approccio limita l'utilità di questi primi database come supporto al programmatore, per i seguenti motivi:

  1. Consentono analisi ed acquisizioni dei dati piuttosto semplici, rispetto a quello che potrebbe essere il campo di applicazione delle reti di sensori.

  2. Come i database tradizionali, sono applicazioni monolitiche che non possono essere facilmente adattabili per il riutilizzo in ambienti imprevisti.

- *Linguaggi di programmazione sofisticati orientati ai segnali*

  l'interfaccia fornita da TinyDB fornisce solo alcune semplici operazioni, adatte ai comuni processi applicativi aziendali. Tuttavia, questo si rivela insufficiente per molte applicazioni di reti di sensori che richiedono linguaggi più sofisticati. (WaveScope and WaveScript Language)

- *Supporto per applicazioni su larga scala*

  I primi sistemi, come TinyDB, hanno ignorato il problema dell'integrazione della rete di sensori all'interno di più vasti ecosistemi di raccolta dati. Ad esempio, sono condotti recenti studi in merito all'integrazione di dati provenienti da diverse fonti di sensori e sulla successiva esecuzione di query di aggregazione su valori ottenuti da diverse insiemi di sensori. (HiFi Project)

- *Supporto per reti mobili e connessioni intermittenti*

Le reti di sensori mobili offrono la potenzialità di monitorare aree geografiche maggiori rispetto alle reti fisse.
Occorre quindi sviluppare una infrastruttura software diversa da quella fornita da TinyDB, dal momento che i nodi mobili possono spesso perdere la connessione e richiedono quindi speciali supporti per una connettività intermittente.
(CarTel Project)

Bibliografia

Siti Internet:

[1] TinyDB: a Declarative Database for Sensor Network
http://telegraph.cs.berkeley.edu/tinydb/

[2] TinyOS
http://www.tinyos.net/

Articoli consultati:

[3] Sam Madden, Michael J. Franklin, Joseph M. Hellerstein and Wei Hong.
TinyDB: An Acqusitional Query Processing System for Sensor Networks.
ACM TODS, 2005

[4] Joseph M. Hellerstein, Wei Hong and Sam Madden.
The Sensor Spectrum: Technology, Trends, and Requirements.
SIGMOD Record 32(4), Dec. 2003

[5] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong.
The Design of an Acquisitional Query Processor for Sensor Networks.
SIGMOD, June 2003.

[6] Sam Madden, Joe Hellerstein, and Wei Hong
TinyDB: In-Network Query Processing in TinyOS

[7] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein
Thinking Big about Tiny Databases

Tesi consultate

Angelo Facondini *“Caratterizzazione sperimentale di reti di sensori basate su IEEE 802.15.4”*

Alessio Cavallini *“Optimization of continuous-monitoring queries in sensor networks”*