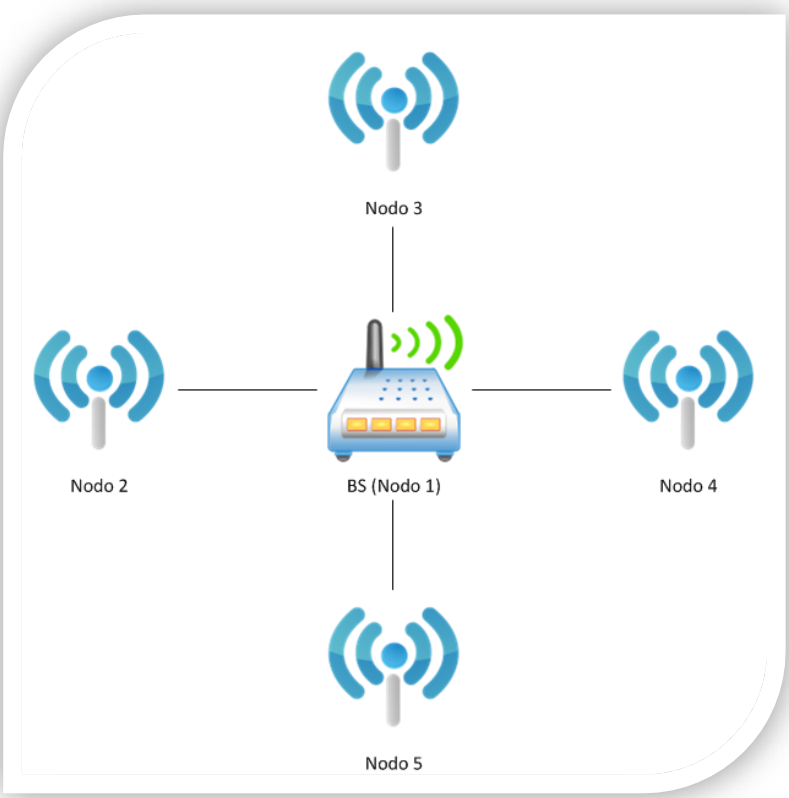
POLLING PROTOCOLL



Corso	Internet of Things
Docente	Matteo Cesana
A.A.	2010-2011
Gruppo di lavoro	Diego Tuzi
	Laerte Saliai
Consegna progetto	05/07/2011

POLITECNICO DI MILANO

SOMMARIO

INTRODUZIONE

POLLING PROTOCOLL

Il progetto Polling Protocoll consiste nello sviluppo di una applicazione in MoteRunner e in TinyOS che costruisca la topologia di rete descritta in figura 1. La Base Station invia periodicamente richieste ai nodi 2, 3, 4 e 5 in modo sequenziale con periodo T_1. I nodi terminali generano dati con periodo $T_2 > T_1$. I nodi terminali devono rispondere alla richiesta della BS con un messaggio speciale se non hanno dati da trasmettere.

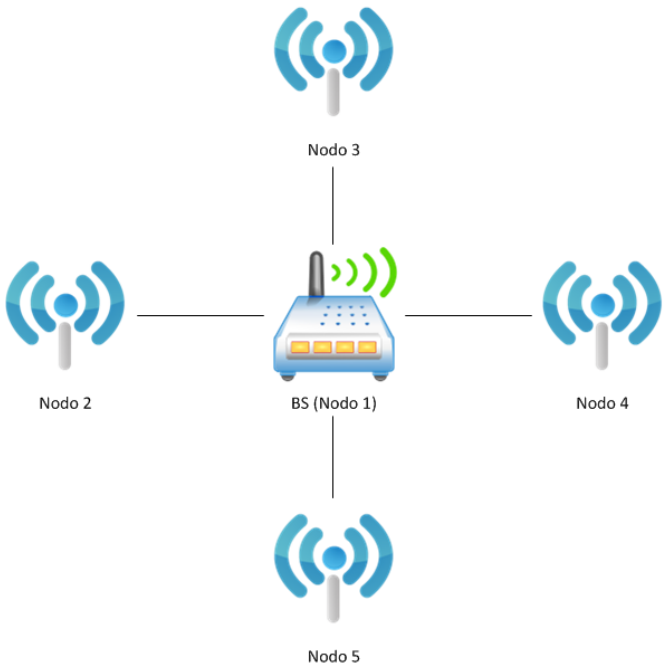


Figura 1 – Topologia di rete.

Un esempio di comunicazione tra BS e nodo terminale è riportata in figura 2.

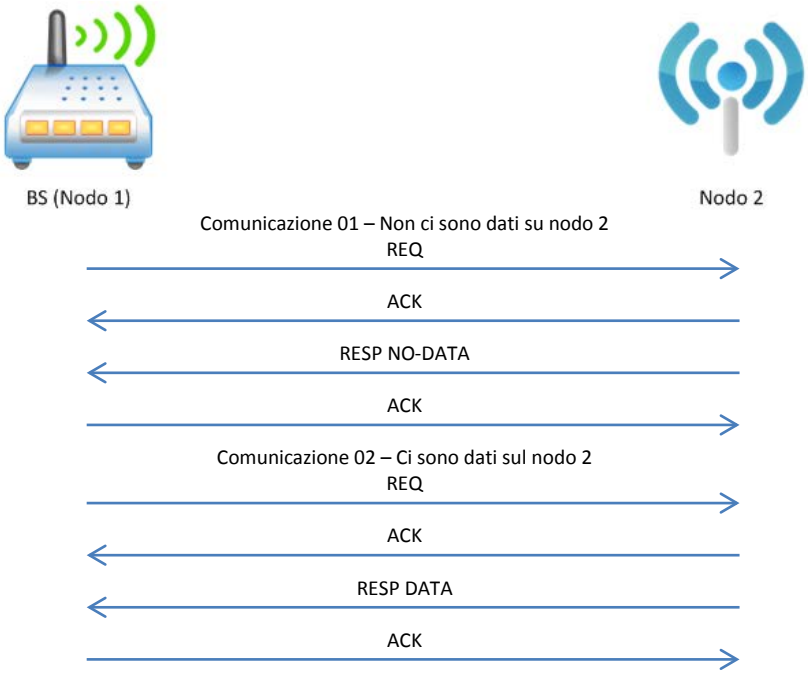


Figura 2 – Esempio di comunicazione tra BS e nodo terminale.

L'applicazione Polling Protocoll implementa inoltre funzionalità di risparmio di energia e di comunicazione visiva tramite l'uso dei leds dei dispositivi.

Il funzionamento del risparmio energetico e dei leds è mostrata in figura 3.

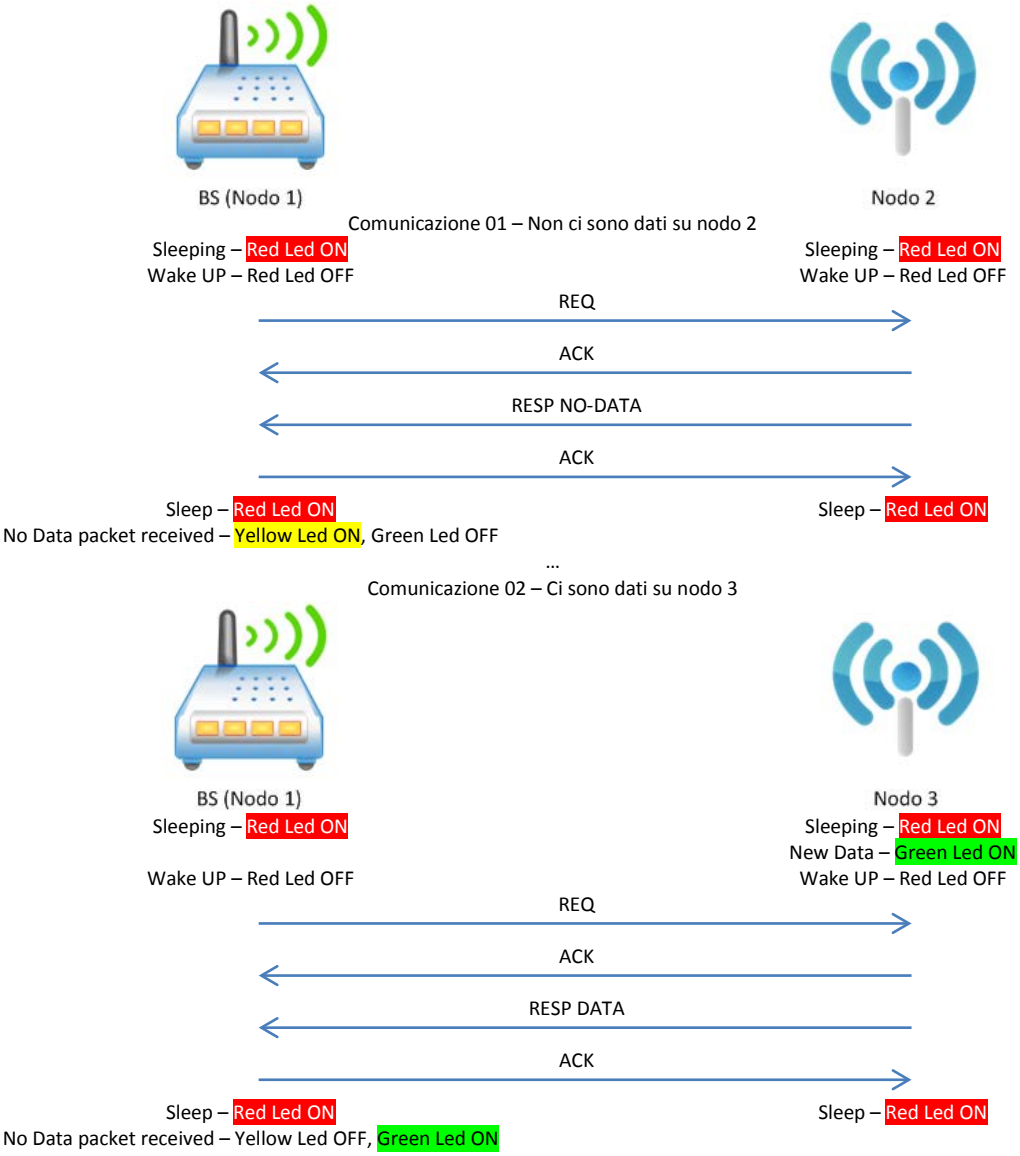


Figura 3 – Esempio di comunicazione tra BS e Nodo terminale con indicazioni sui Leds e delle operazioni di risparmio energetico.

Nel progetto si sono utilizzati i seguenti parametri: $T_1 = 1000\ ms$ e $T_2 = 6000\ ms$ o $T_2 = 5000\ ms$. I meccanismi di risparmio energetico permettono di avere le seguenti espressioni:

- Duty cycle della Base Station: $duty_{bs} = \frac{T_{active}}{T_{active}+T_{inactive}} = \frac{300\ ms}{300\ ms\ +700\ ms} = 0.3$;
- Duty cycle del nodo terminale: $duty_{terminal} = \frac{T_{active}}{T_{active}+T_{inactive}} = \frac{300\ ms}{300\ ms\ +3700\ ms} = 0.075$.

I valori di sopra possono essere ulteriormente ridotti eseguendo un'accurata analisi dell'effettivo tempo di utilizzo della parte radio. Nel progetto si è preferito mantenere una configurazione molto prudente.

POLLING PROTOCOLL – MOTERUNNER

INSTALLAZIONE

L'installazione di MoteRunner richiede la versione 1.6 di JDK e broswers recenti come Firefox 3.5 o Internet Explorer 8.0. Il pacchetto d'installazione è reperibile dal sito http://www.zurich.ibm.com/moterunner. Esistono procedure guidate per l'installazione in diversi sistemi operativi. Per lo sviluppo del progetto si è scelta la versione per Windows.

AMBIENTE DI SVILUPPO

Lo sviluppo di un'applicazione MoteRunner richiede la scrittura di files sorgenti in linguaggio Java o C#. L'installazione fornisce tutto il necessario per sviluppare e testare il codice. In particolare è fornito un IDE basato su Eclipse e una dashboard utilizzabile da linea di comando o tramite browser. Prima di avviare la simulazione è necessario che i files sorgenti .java o .cs che contengono il codice siano compilati. Per avviare la simulazione è necessario digitare dal prompt dei comandi o da una shell il comando "mrsh". A tal punto si può proseguire tramite linea di comando o più semplicemente aprire il browser ed andare all'indirizzo: http://localhost:5000/comote/html/dashboard.html. Dalla pagina web è possibile fare tutte le azioni necessarie per la simulazione: creazione dei motes, caricamento dei programmi sui motes, topologia di rete tramite net grid, visualizzazione delle informazioni scambiate in rete tra motes e molto altro.

SVILUPPO DEL PROGETTO

Il progetto PollingProcotoll è stato sviluppato utilizzando vari esempi presenti all'interno dell'installazione base di Moterunner. Il principale è codice sorgente cui ci si è riferiti, è Blink.java.

APPLICAZIONE MOTERUNNER

Il codice sorgente dell'applicazione PollingProtocoll si compone dei seguenti files:

- PollingPro.java;
- PollingNei.java.

Il file PollingPro.java rappresenta il codice sorgente del programma da installare nel mote che avrà funzione di "Base Station" mentre PollingNei.java è il codice sorgente del programma da installare nei mote "Neighborgs".

La divisione in due programmi differenti permette un risparmio in memoria e l'esecuzione di un minor numero d'istruzioni.

Il profilo Base Station ha le seguenti caratteristiche:

- Invio periodico di richieste ai neighborgs in modo ciclico e con periodo T_1. I messaggi non sono di tipo broadcast ma contengono l'indirizzo di destinazione del neighborg destinatario.
- Ricezione di messaggi di tipo Data o no-Data.
- Gestione di callbacks per lo spegnimento/accensione della parte radio.

Il profilo Nodo Neighborg ha le seguenti caratteristiche:

- Generazione di dati casuali con periodo $T_2 > T_1$.
- Invio di risposte di tipo DATA o NO-DATA. Nel caso di RESP di tipo DATA, il dato del pacchetto è un dato casuale.
- Gestione di callbacks per lo spegnimento/accensione della parte radio.

SIMULAZIONE CON MOTERUNNER

Per compilare i file PollingPro.java e PollingNei.java si usa il comando: “mrc --assembly=pollingPro-x.x PollingPro.java -r logger-x.x” (ove x.x indica la versione). Per eseguire la simulazione, con uno scenario di rete già configurato, si può utilizzare lo script “pollingScript.mrsh”. La chiamata dello script si esegue con il comando “mrsh pollingScript.mrsh”. Si ricorda che può rivelarsi necessario ricompilare i sorgenti per eseguire la simulazione su pc diversi.

Nel caso di configurazione manuale della rete, è necessario avviare prima il mote BS cosicché si metta in ascolto dei neighborgs.

L’ascolto del canale si può eseguire tramite la scheda Control Panel >> Radio come mostrato in figura 4.

Per il progetto, si è scelto di visualizzare le informazioni solo sul canale Info, visibile nella scheda Console >> Info.

In figura 4 si fornisce uno screenshot della dashboard di mote runner durante la simulazione del progetto Polling Protocoll.

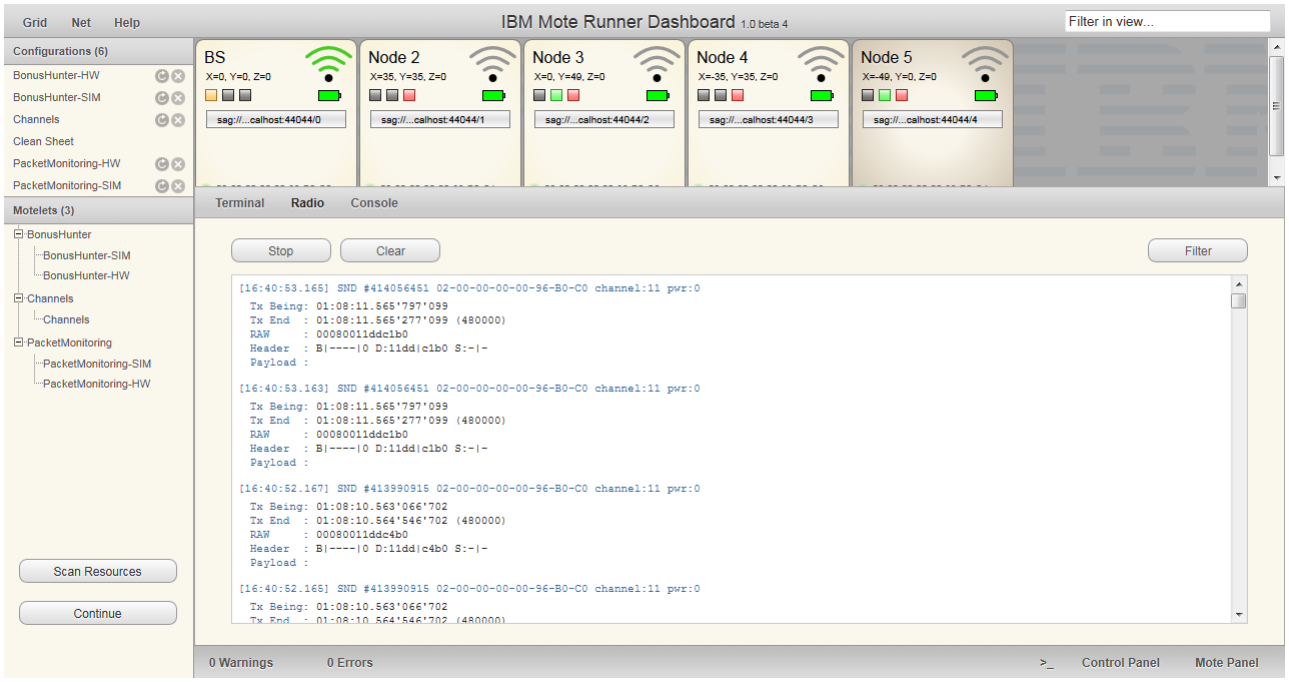


Figura 4 – Dati scambiati tra BS e neighborgs .

In figura 5 si mostra la scheda Net in cui è definita una particolare configurazione di rete.

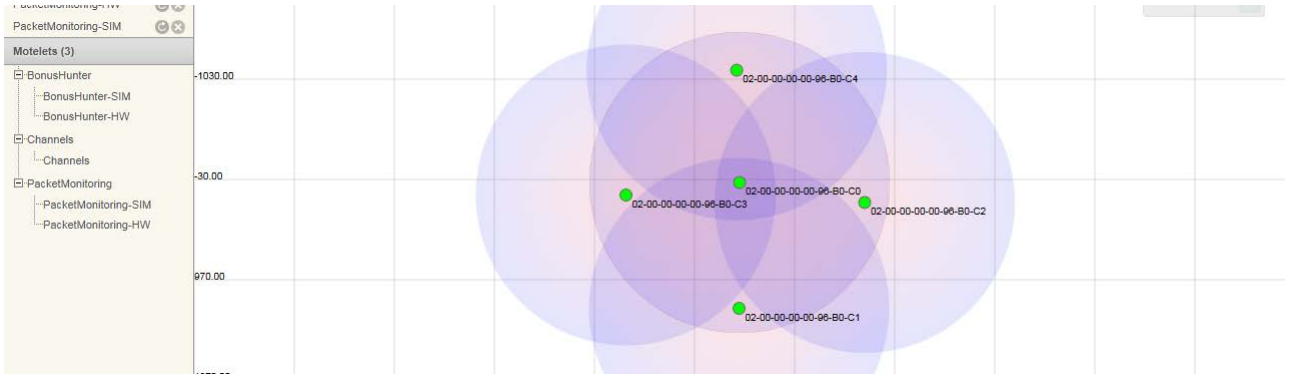


Figura 5 – Visualizzazione griglia di comunicazione tra BS e neighborgs.

POLLING PROTOCOLL – TINYOS

INSTALLAZIONE

Per l'installazione di tinyOS 2.1.1 si è seguita la wiki ufficiale del progetto TinyOS disponibile all'indirizzo http://docs.tinyos.net/. Si è utilizzato il sistema operativo Ubuntu, tale scelta è stata guidata dalla presenza di repository tinyOS per l'istallazione automatica. La guida ufficiale non descrive il procedimento d'installazione su versioni di s.o. recenti, è comunque possibile reperire sull'argomento una grande quantità di materiale su Internet.

AMBIENTE DI SVILUPPO

Lo sviluppo di un'applicazione tinyOS richiede la scrittura di diversi tipi di file, il linguaggio utilizzato è il **nesC**. La simulazione è effettuata con l'ausilio di TOSSIM. Per lanciare la simulazione è necessario un file in **Python** (o in C++).

Per agevolare lo sviluppo si è utilizzato un IDE, ECLIPSE. Di norma ECLIPSE non supporta files nesC e Python, tuttavia sono presenti dei plug-ins che permetto numerose funzionalità, quali: error detection, code completion, debug, outline e altro. In particolare per nesC si è utilizzato YETI 2 (http://tos-ide.ethz.ch/), mentre per python si è utilizzato PYDEV (link http://pydev.org/).

SVILUPPO DEL PROGETTO

Il progetto PollingProcotoll è stato sviluppato a partire dal codice dell'esempio "SendAckExample" messo a disposizione sul sito del corso di IoT.

APPLICAZIONE TINYOS

Il codice sorgente dell'applicazione PollingProtocoll si compone dei seguenti files:

- PollingProtocollC.nc;
- PollingProtocollAppC.nc;
- FakeSencorC.nc;
- FakeSencorP.nc.

Il file più importante è costituito dal modulo PollingProtocollC.nc, che racchiude l'implementazione del codice sorgente dell'intera applicazione. Gli altri files sono di supporto al modulo principale.

All'interno del modulo PollingProtocollC.nc possono essere distinti due profili, Base Station e Nodo Terminale.

Il profilo Base Station ha le seguenti caratteristiche:

- "MilliTimer", per l'invio periodico di messaggi di REQ, con periodo T_1 e con campo *destination* che varia ciclicamente da 2 a 5;
- Task "sendReq()", che implementa l'invio del messaggio di REQ.

Il profilo Nodo Terminale ha le seguenti caratteristiche:

- " DataTimer", per pianificare la generazione di dati con periodo $T_2 > T_1$;
- Task "sendRespData()", per l'invio di risposte di tipo DATA, nel quale si richiama" Read" per la lettura dei dati su un falso sensore;
- Task "sendRespNoData()", per l'invio di risposte di tipo NO-DATA.

Inoltre i due profili condividono l'implementazione delle seguenti funzionalità:

- "Send", per l'invio pacchetti;
- "Receive", per la ricezione di pacchetti;
- "SleepTimer", per lo spegnimento della parte radio;
- "WakeUpTimer", per l'accensione della parte radio.
- Segnalazione visiva tramite leds.

La generazione dei dati è fornita dai files FakeSencorC.nc e FakeSensorP.nc che simulano la lettura dei dati da un falso sensore.

SIMULAZIONE CON TOSSIM

Per simulare l'applicazione devono essere eseguite le seguenti operazioni:

- Compilazione;
- Configurazione della simulazione in Python (o C++).

La compilazione è molto semplice da eseguire. Dopo avere creato il Makefile, basta lanciare da shell il comando *make micaz sim*. La compilazione genera i files necessari per lanciare la simulazione.

La configurazione della simulazione consiste nella realizzazione di un file di script che può essere in Python o in C++. Per il progetto si è creato lo script "RunSimulationScript.py".

"RunSimulationScript.py" richiama i seguenti files:

- Files derivanti dalla compilazione (ad esempio TOSSIM.py);
- mayer-heavy.txt;
- topology.txt.

Lo script "RunSimulationScript.py" si occupa dei seguenti compiti:

- Inizializzazione dei componenti necessari alla simulazione (nodi, radio layer, mac layer);
- Creazione del canale radio per i nodi della rete utilizzando topology.txt;
- Creazione del modello di canale CPM utilizzando mayer-heavy.txt;
- Creazione dei canali di debug, necessari per avere informazioni sugli eventi che occorrono;
- Start della simulazione, iterando il metodo runNextEvent();
- Generazione di output su terminale o su file di testo "simulation.txt".

CONCLUSIONI – MOTERUNNER VS TINYOS

Si espongono le principali differenze riscontrate durante la realizzazione del progetto.

	Mote Runner	TinyOS
Installazione	Non sono state riscontrate particolari differenze. In entrambi i casi esistono procedure automatiche per portare a termine la procedura d'installazione.	
Ambiente di sviluppo	Fornisce un ambiente si sviluppo completo: Mote Runner IDE per la scrittura del codice e Mote Runner DashBoard per simulazioni grafiche in tempo reale.	L'installazione di per se non fornisce un ambiente completo. Sono presenti su internet tools che permettono di migliorare quest'aspetto.
Linguaggio di programmazione	Si possono utilizzare sia Java sia C#. Nel progetto si è scelto di utilizzare Java. Comunque, in entrambi i casi si dispone di un linguaggio più evoluto e facile da comprendere.	Si utilizza NesC che è simile al C, e risulta un linguaggio meno intuitivo e più formale.
Sorgenti	Si possono realizzare applicazioni mediante l'utilizzo di un solo file sorgente .java o .cs.	Si utilizzano necessariamente più files sorgenti, header, configurazioni e moduli.
Compilazione	Se il sistema è correttamente configurato la compilazione richiede un semplice comando: "mrc –assembly=nomeProgramma-x.x nomeProgramma.java". Si nota che qualora si utilizzassero ulteriori librerie oltre a quella standard, esse devono essere include nella compilazione mediante il comando "-r nomeLibreria-x.x". Per il progetto è necessario includere "logger-x.x".	È necessario creare un Makefile. A tal punto se il sistema è correttamente configurato, il comando "make platform" genera l'applicativo da installare sul mote, mentre il comando "make micaz sim" genera i file necessari per la simulazione.
Simulazione	La simulazione è molto semplice e intuitiva. Ottenuti i files risultanti dalla compilazione, è possibile caricarli sui motes virtuali semplicemente con un clic. Si ha a disposizione un ambiente grafico notevolmente facile da utilizzare e ricco di funzionalità. La gestione del canale radio tra i nodi si esegue tramite l'uso di una griglia 2d, nella quale è possibile configurare una topologia di rete qualsiasi. Inoltre la simulazione è in tempo reale e si può vedere l'avanzamento dell'applicazione attraverso i led sui mote, attraverso l'ascolto del canale radio oppure osservando i canali per le info, warning, debug, ecc.	La simulazione con Tossim è molto potente, l'unico ostacolo è la creazione dello script (Python o C++) che avvia la simulazione. In tale script è necessario creare la topologia e creare il modello di canale, tutti aspetti che in tinyOs sono trasparenti all'utente. Inoltre per la simulazione non sono supportate molte piattaforme.
Log	Si ha a disposizione un numero limitato di canali su cui visualizzare informazioni sulla simulazione.	Il numero di canali su cui visualizzare informazioni sulla simulazione non è limitato.
Documentazione	Esiste una documentazione abbastanza completa fornita da IBM. Tuttavia, essendo Mote Runner una piattaforma recente, non è facile recuperare materiale da altre fonti.	Esiste una documentazione abbastanza completa fornita dagli sviluppatori del progetto. Qualora la documentazione ufficiale non fosse esaustiva, sulla rete è presente un grande quantitativo di materiale, forum e quanto altro che sopperiscono a tale mancanza.

Si precisa che quanto sopra è un commento puramente personale maturato sulla realizzazione del progetto e relativo solo agli aspetti che si sono affrontati.