Progetto di RHS
MicroAODV per Reti di Sensori
A.A. 2007/2008

Si consideri una rete di sensori MicaZ con sistema operativo TinyOS, dove ogni nodo è identificato da un ID unico e dove è presente un solo nodo sink. La rete non ha vincoli riguardo la mobilità dei nodi, in altri termini i nodi possono essere statici o mobili.

La rete di sensori è controllata dal sink ed utilizza una versione semplificata del protocollo di routing AODV per permettere al sink di comunicare direttamente con un qualsiasi sensore tramite messaggi unicast. Il protocollo di routing non prevde la comunicazione tra sensori arbitrari, ma solo tra un sensore e il sink o viceversa.

Strutture dati e formato pacchetti

Tabella di rotta

Ai fini del routing ogni sensore mantiene una tabella di rotta i cui elementi hanno i seguenti campi:

`<VALIDITY, IDDest, IDNexthop, Distance, Expiration, RREQ_PENDING>`

Dove:

- `VALIDITY`: indica se le informazioni di rotta contenute nell'elemento sono valide (in tal caso contiene il valore `VALIDITY=TRUE`);
- `IDDest`: è l'identificatore del nodo destinazione per il quale l'elemento della tabella conserva la rotta;
- `IDNexthop`: è l'ID del nodo al quale vanno inoltrati i pacchetti indirizzati a `IDDest`;
- `Distance`: è la distanza in hop dal nodo `IDDest`;
- `Expiration`: è il tempo di scadenza dell'elemento. Scaduto questo tempo l'elemento viene marcato come non valido. Ogni volta che l'elemento della tabella viene creato o aggiornato questo campo viene reinizializzato ad una costante `STD_EXPIRATION`.
- `RREQ_PENDING`: Questo campo è usato SOLO dal sink. Se `RREQ_PENDING=True` significa che il sink ha già inviato un pacchetto di `RREQ` per la destinazione `IDDest`, ma non ha ancora ricevuto il corrispondente messaggio di `RREP`.

Formato dei pacchetti

In generale un pacchetto MAC ha un formato di questo tipo:

```
<IDSource, IDDest, MACPayload>
```

Dove `IDSource` è il'`ID` del nodo che spedisce il pacchetto, `IDDest` è l'`ID` del nodo al quale il pacchetto deve essere inviato, e `MACPayload` è un pacchetto di livello rete il cui formato dipende dal tipo di pacchetto. A livello di rete MicroAODV definisce tre tipi di pacchetti: `DATA`, `RREQ`, `RREP`.

Il pacchetto di tipo `RREQ` è usato per la costruzione delle rotte e viene sempre generato dal sink. Il formato è il seguente:

```
<IDSource=sink, TYPE=RREQ, RREQID, IDDest, HopCount>
```

Dove:

- IDSource indica l'ID del nodo che ha generato il pacchetto. Come verrà discusso nel seguito, in microAODV il mittente è sempre il sink.
- TYPE=RREQ indica che si tratta di un pacchetto di Route Request.
- RREQID è un identificatore univoco dei pacchetti di Route Request. E' un numero intero (a 8 bit) che viene stabilito dal sink (i dettagli sono riportati nella sezione Route Request).
- IDDest è l'ID del nodo per il quale si esegue il protocollo di Route Discovery, ovvero è il nodo per il quale il sink vuole individuare una rotta.
- HopCount un intero indica quanti hop ha attraversato (i dettagli sono riportati nella sezione Route Request).

Il pacchetto di tipo RREP è usato nel protocollo di route discovery per rispondere ai pacchetti di RREQ e ha il seguente formato:

```
<IDSource, TYPE=RREP, IDDest=sink, HopDist, ResHopDist>
```

Dove:

- IDSource indica l'ID del nodo che ha generato il pacchetto.
- TYPE=RREP indica che si tratta di un pacchetto di Route Reply.
- IDDest è l'ID del nodo al quale il pacchetto RREP va inviato. Come verrà discusso nel seguito, nel nostro caso il destinatario è sempre il sink.
- HopDist è la distanza tra il nodo IDSource e il nodo IDDest (i dettagli sono riportati nella sezione Route Reply).
- ResHopDist è distanza tra il nodo IDSource e il nodo che inoltra il pacchetto (i dettagli sono riportati nella sezione Route Reply)

Il pacchetto di tipo DATA è usato per spedire dati e ha il seguente formato:

```
<IDSource, TYPE=DATA, IDDest, Payload>
```

Dove:

- IDSource indica l'ID del nodo che ha generato il pacchetto
- TYPE=Data indica che si tratta di un pacchetto dati
- IDDest è l'ID del nodo al quale il pacchetto dati va inviato
- Payload è il contenuto del messaggio.

Buffer di spedizione dei pacchetti

Quando un nodo deve spedire un pacchetto dati ma non conosce la rotta per la destinazione bufferizza temporaneamente il pacchetto in un buffer di spedizione. Questo buffer è un array di MAXBUF elementi che hanno il seguente formato:

```
<Validity, Destination, Message>
```

Dove Validity=True se l'elemento contiene un pacchetto valido da spedire.
Il campo Destination indica l'ID del nodo al quale il campo Messaggio va spedito. In microAODV per ogni nodo diverso dal sink il valore del campo Destination è sempre sink.

Invio di pacchetti dal sink ai nodi

Quando il sink vuole spedire un pacchetto P al sensore x, verifica per prima cosa se ha nella tabella un'elemento valido <True,idd,idn,d,e,RREQ_PENDING> tale che idd=x.
Se l'elemento esiste e RREQ_PENDING=False inoltra il pacchetto P al nodo idn.

Se l'elemento esiste ma RREQ_PENDING=True allora bufferizza il pacchetto P finchè non riceve il messaggio di RREP corrispondente.
Altrimenti il sink bufferizza il pacchetto P, crea nella tabella di rotta un elemento <VALIDITY=True, x, -, -, -, RREQ_PENDING=True> ed esegue il protocollo di route discovery per individuare un percorso per raggiungere x.

Route discovery

Il protocollo di route discovery per il nodo x iniziato dal sink invia in broadcast locale un pacchetto di Route Request RREQ che contiene i seguenti campi:

<IDSource=sink, RREQ, RREQID, IDDest, HopCount>

dove:

- IDSource: ID del mittente (ovvero ID del sink);
- RREQ: Un codice che identifica il pacchetto come una RREQ;
- RREQID: un identificatore di Route Request che viene incrementato dal sink ogni volta che emette un nuovo pacchetto di RREQ;
- IDDest: L'ID del nodo per il quale va ricercata la rotta (in questo caso è x)
- HopCount: Un contatore di hop inizializzato a 0.

Ogni volta che un nodo y riceve un pacchetto di RREQ <sink,RREQ,r,x,h> da un nodo z opera nel modo seguente:
Se y ha già ricevuto un pacchetto di RREQ con identificatore maggiore o uguale ad r allora scarta il pacchetto (che è già stato ricevuto e inoltrato).
Altrimenti:

1. Copia il valore r in una variable interna URREQID che contiene l'ultimo identificatore di RREQ che ha ricevuto.
2. Controlla nella sua tabella di rotta se esiste un elemento valido per il sink e questa entry ha un campo Distance di valore d>h allora aggiorna l'elemento ponendo i campi IDNexthop=z,Distance=h e Expiration=STDEXPITARION.
3. Se nella sua tabella di rotta NON esiste alcun elemento valido per il sink allora viene creato l'elemento: <True,sink,z,h,STDEXPITARION>
4. Se y=x invia al sink un pacchetto di Route Reply RREP=<sink,RREP,y,h,h1=0> al sink (il campo h1 serve per contare gli hop dal nodo y al sink); il pacchetto verrà inoltrato usando le informazioni presenti nella tabella di rotta.
5. Altrimenti inoltra il pacchetto di RREQ a tutti i suoi vicini con il numero di hop incrementato di 1, ovvero:
   <sink,RREQ,r,x,h+1>

Route reply

Il pacchetto di RREP si propaga verso il sink in unicast. In particolare, ogni volta che un nodo y riceve da un nodo z un pacchetto di RREP=<x, RREP, sink, h, h1> opera nel modo seguente:

1. Crea (o aggiorna) un elemento <True,x,z,h1,STDEXPITARION> nella sua tabella di rotta per il nodo x.
2. Incrementa h1 e inoltra il pacchetto di RREP verso il sink.

Quando il sink riceve un pacchetto RREP=<x,RREP,sink,h,h1> dal nodo z aggiorna la sua tabella di rotta con la entry:

<VALIDITY=True, x, z, h, STDEXPITARION,RREQ_PENDING=False>

E quindi spedisce al nodo x tutti i pacchetti che sono stati bufferizzati in precedenza, incluso il pacchetto P (nota che è possibile aver bufferizzato più pacchetti da spedire per la stessa destinazione).

Routing

Quando un nodo y riceve un pacchetto dati: <IDSource, TYPE=DATA, IDDest=x, P> da inoltrare al nodo x effettua le seguenti operazioni:

- Se la tabella di rotta di y contiene un elemento <VALIDITY=True, x, z, Distance, Expiration, RREQ_PENDING=False> allora y inoltra <x,P> verso z.
- Se y=sink e la sua tabella di rotta contiene un elemento <VALIDITY=True, x, z, Distance, Expiration, RREQ_PENDING=True> il pacchetto viene bufferizzato (se c'è spazio nel buffer, altrimenti viene scartato) per essere spedito non appena RREQ_PENDING diventa False.
- Se il pacchetto è diretto al sink ma il mittente non contiene un elemento nella tabella di rotta valido per il sink allora bufferizza il pacchetto in attesa che il sink esegua una route discovery (che ha l'effetto indiretto di individuare un percorso verso il sink da qualsiasi nodo della rete).
- Altrimenti, se la tabella di rotta non contiene alcun elemento valido per il nodo x allora il pacchetto viene scartato senza riportare alcun errore al sink.

Eliminazione elementi dalle tabelle di rotta

Quando scade il timer di "expiration" di un elemento della tabella di rotta <VALIDITY, IDDest, IDNexthop, Distance, Expiration, RREQ_PENDING>, ci sono due casi:

- Se scade in un nodo generico l'elemento viene marcato come non valido.
- Se scade nel sink ci sono due sottocasi:
    - Se RREQ_PENDING=False allora l'elemento viene marcato come non valido
    - Altrimenti il sink genera un nuovo pacchetto di RREQ per la destinazione IDDest.

Note

Nota1: Nel modello di comunicazione previsto nella rete si assume che ogni comunicazione sia iniziata dal sink, per cui solo il sink esegue il protocollo di route discovery. Ogni altro nodo che vuole comunicare (e può farlo solo col sink) o ha un elemento nella tabella di rotta che individua un cammino verso il sink, oppure deve attendere che il sink effettui una RREQ (perché in questo modo tutti i sensori inizializzano un elemento nella loro tabella di rotta verso il sink).

Nota 2: Non si richiede di gestire perdite di pacchetti e acknowledgement, né si richiede di gestire i timeout nei pacchetti di RREQ.
Questo significa che se un pacchetto dati viene inoltrato su una rotta non più valida (ad esempio perché la destinazione si è spostata) il pacchetto viene perso.
Inoltre, se un pacchetto di RREQ viene perso è possibile che non si possa inoltrare un pacchetto dati, ma allo scadere del timeout Expiration dell'elemento corrispondente nella tabella di routing il sink genera nuovamente un pacchetto di RREQ per costruire una rotta.

Assegnamento

Si richiede di implementare un programma Java che in un ciclo infinito esegue le seguenti operazioni:

1. legge da tastiera un ID `x` di un sensore,
2. invia un pacchetto al sensore `x` con la richiesta di lettura del valore attuale della luce. Il pacchetto è inoltrato al sensore `x` usando il protocollo di routing sopra descritto.
3. quando riceve dal sensore `x` il dato di luce lo visualizza.

Quando il sensore `x` riceve la richiesta legge la luce dal suo trasduttore `phototemp` e invia un pacchetto col valore letto al sink. Il dato di luce è quindi inoltrato dal sink all'applicazione Java che lo visualizza.

In particolare è necessario spedire un file in formato .zip o .tar.gz (SOLO questi formati sono accettati) all'indirizzo ste@di.unipi.it contenente:

- una versione stampabile del codice (adeguatamente commentato) in un unico file in formato .pdf.
- una relazione che descriva la struttura del codice e le scelte implementative e che spieghi come compilare e testare il codice.
- i file sorgenti.

Altre informazioni e documentazione sono disponibili nella pagina web del corso:
http://www.cli.di.unipi.it/doku/doku.php/rhs/rhs_2007-2008

Suggerimento: il codice del simulatore è in grado di simulare solo nodi identici, quindi il codice del sink deve essere uguale a quello degli altri nodi. Pertanto si consiglia di scrivere un unico codice che include tutte le funzionalità (del sink o del nodo) che seleziona la porzione del codice da eseguire in funzione dell'ID del nodo sul quale viene eseguito. Il codice scritto in questo modo è eseguibile sia sul simulatore sia sui sensori reali.