Università degli studi di Catania

Corso di Laura Magistrale Informatica

P2P & Wireless networks A.A 2014-2015

Antonio Fischetti W82000021

Implementazione in ns2 del DTN (Delay Tolerant Network) nel protocollo DSR

Introduzione

Lo scopo del progetto è stato quello di implementare in ns2 il **DTN (Delay Tolerant Network)** modificando il protocollo DSR (*Dynamic Source Routing*), protocollo di routing utilizzato nelle Mobile ad Hoc Networks.

Il **Delay Tolerant Network** rappresenta un'architettura di rete di telecomunicazione utilizzata in reti caratterizzate da ritardi nella trasmissione. Questo si basa su un meccanismo di *store and forward message*. Ovvero, se durante la trasmissione di uno o più pacchetti da un nodo sorgente ad un nodo destinazione, qualcosa nella comunicazione fallisce, il nodo trasmittente memorizza il pacchetto per poi spedirlo nuovamente al nodo destinazione.

Quanto sopra citato è l'obiettivo del progetto, realizzato attraverso una modifica al DSR. Nello specifico durante lo scambio di pacchetti tra i nodi di una rete wireless simulata attraverso ns2, alcuni di questi non riescono ad arrivare a destinazione. Per questo motivo, utilizzando una struttura dati (una coda) memorizzo i pacchetti per poi provare a rispedirli successivamente.

Implementazione

Per lo sviluppo del protocollo DTN sono state effettuate alcune modifiche alla classe DSRAgent:

1. Creazione di una struttura dati (coda) contenente i pacchetti da rinviare.
2. Controllo e salvataggio dei pacchetti nel metodo DSRAgent::xmitFailed.
3. Creazione del metodo RecallMethod::expire richiamato ad intervalli di tempo.
4. Controllo di avvenuta ricezione pacchetto nel metodo DSRAgent::recv.

Creazione struttura dati (coda) contenente i pacchetti da rinviare.

La struttura dati creata è una coda. Questa ha il compito di contenere al suo interno tutti i pacchetti che non sono stati trasmessi da un nodo sorgente ad un nodo di destinazione e che successivamente vengono rispediti. La scelta di questa struttura dati nasce dalla gestione di tipo FiFo (First Input First Output), quindi i pacchetti persi per prima, sono quelli che per prima vengono rinviati.

Controllo e salvataggio dei pacchetti nel metodo DSRAgent::xmitFailed.

xmitFailed è un metodo di callback che viene richiamato quando un pacchetto non raggiunge la sua destinazione, con la successiva propagazione di un pacchetto di errore verso il nodo sorgente. Al suo interno transitano diversi tipi di pacchetti (dati, request, error), quindi effettuando una serie di controlli scarto quelli che non sono necessari e salvando quelli dati all'interno della struttura dati attraverso l'utilizzo di una **push_back**.

1. Controllo il tipo di pacchetto *cmh->ptype() != PT_DSR*
2. Controllo che il pacchetto non sia nullo *cmh->uid() != 0*
3. Controllo address corrente-successivo srh->get_next_addr() != srh->cur_addr()

```
if(srh->get_next_addr() != srh->cur_addr() && cmh->ptype() != PT_DSR && cmh->uid() != 0 )
   {

        pacchetti_da_reinviare.push_back(pkt->copy());
        cout  << "UID_xmitFailed= " <<cmh->uid() <<" C_Add=" <<srh->cur_addr() <<" N_Add=" <<srh->get_next_addr()
              <<" T=" <<cmh->ptype() <<" Time=" <<Scheduler::instance().clock() <<" Dump=" << srh->dump() <<" Add="
              << srh->addrs()  << endl << flush;

   }
```

Creazione del metodo RecallMethod::expire richiamato ad intervalli di tempo.

Ho implementato questo metodo per permettere l'invio dei pacchetti che si trovano all'interno della coda ad intervalli di tempo. Per prima cosa viene controllato che la struttura dati non sia vuota ed in caso positivo viene richiamato in modo ciclico il metodo per l'invio dei pacchetti **sendOutPacketWithRoute** e successivamente viene effettuata una pop (estrazione) dalla coda.
Sempre all'interno di questo metodo , attraverso un ciclo for, effettuo una stampa nella console della quantità di energia consumata fino a quell'istante ad ogni nodo. (l'informazione è necessaria per effettuare un confronto tra il DSR ed il DSR con il protocollo DTN implementato).

```
void
RecallMethod::expire(Event *)
{

if (pacchetti_da_reinviare.size() > 0)
{
        while (pacchetti_da_reinviare.size() > 0)
        {

           hdr_sr *srh = hdr_sr::access(pacchetti_da_reinviare.front());
           hdr_ip *iph = hdr_ip::access(pacchetti_da_reinviare.front());
           hdr_cmn *cmh = hdr_cmn::access(pacchetti_da_reinviare.front());

           SRPacket p(pacchetti_da_reinviare.front(), srh);

 cout  << " Size_Temp_Coda= " << pacchetti_da_reinviare.size() <<" UID=" << cmh->uid() <<" C_Add=
<<srh->get_next_addr() <<" " << "Time=" << Scheduler::instance().clock()  << endl << flush;
        if(cmh->uid() !=0 ) //Nb quelli reinviati cancellati dalla coda diventano con cmh->uid()
         a_->sendOutPacketWithRoute(p, true, 0.3);
          pacchetti_da_reinviare.pop_front();

        }

}
else
{
cout  << "Coda Vuota al Tempo =" << Scheduler::instance().clock() << endl << flush;
}

cout << "T=" << Scheduler::instance().clock() <<" Energy" <<flush;
int num_nodes = God::instance()->nodes();
for ( int i = 0 ; i< num_nodes ; i++)
{
a_->iNode[i] = (MobileNode *)(Node::get_node_by_address(i));
a_->iEnergy[i] = a_->iNode[i]->energy_model()->energy();


printf(" N%i=%.4f",i,a_->iEnergy[i]);
}
printf("\n");


// Richiama il metodo dopo un totale di secondi decisi
  resched(CHIAMA_RECALL);


}
```

Controllo di avvenuta ricezione pacchetto nel metodo DSRAgent::recv.

Questo metodo viene richiamato quando un pacchetto arriva al nodo destinatario, quindi assicura l'avvenuta ricezione. Il controllo che effettuo all'interno è quello di verificare se un pacchetto che non era arrivato a destinazione e quindi salvato nella coda, successivamente viene ricevuto. Modifico quindi l'uid (identificativo del pacchetto) settandolo a 0, in modo che durante il controllo precedente al rinvio dei pacchetti dati persi, quelli settati a 0 verranno scartati della coda senza rinvio.

```
if (pacchetti_da_reinviare.size() > 0)
{

  int sizePack = pacchetti_da_reinviare.size();

for (int i = 0 ; i < sizePack ; i++)
{
            hdr_sr *srhTMP = hdr_sr::access(pacchetti_da_reinviare.at(i));
            hdr_ip *iphTMP = hdr_ip::access(pacchetti_da_reinviare.at(i));
            hdr_cmn *cmhTMP = hdr_cmn::access(pacchetti_da_reinviare.at(i));

            SRPacket pTMP(pacchetti_da_reinviare.at(i), srhTMP);


if (cmhTMP->uid() == cmh->uid() && srh->dump() == srhTMP->dump() )
        {

cout  << "UID_Pack_Recv= " <<cmhTMP->uid() <<" CurAdd=" <<srhTMP->cur_addr() <<" NextAdd="
<<cmhTMP->ptype() <<" Time=" <<Scheduler::instance().clock() <<" Add=" << srh->addrs()  <<

                 cmhTMP->uid() = 0;
                return;

        }

        pTMP.pkt = NULL;

}
}
```

Simulazione

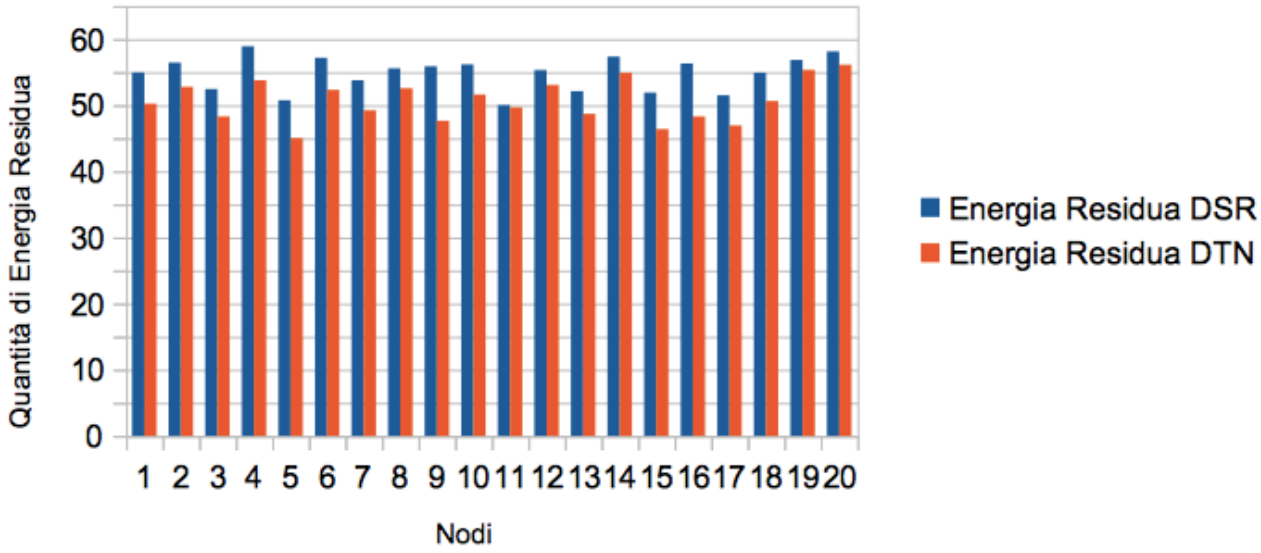
Per constatare le differenze tra l'algoritmo DSR e quello modificato con l'aggiunta del DTN è stato creato un ambiente wireless con le seguenti caratteristiche:

Numero nodi	20
Dimensione Griglia	1000*1000
Movimento	Random
Tipo traffico	CBR
Numero massimo connessioni	10
Dimensione pacchetti	512 bytes
Tempo di simulazione	1000 sec
Consumo rxPower	0,1 watt
Consumo txPower	0,3 watt
Consumo sleepPower	0,02 watt
Energia Iniziale nodi	100 watt

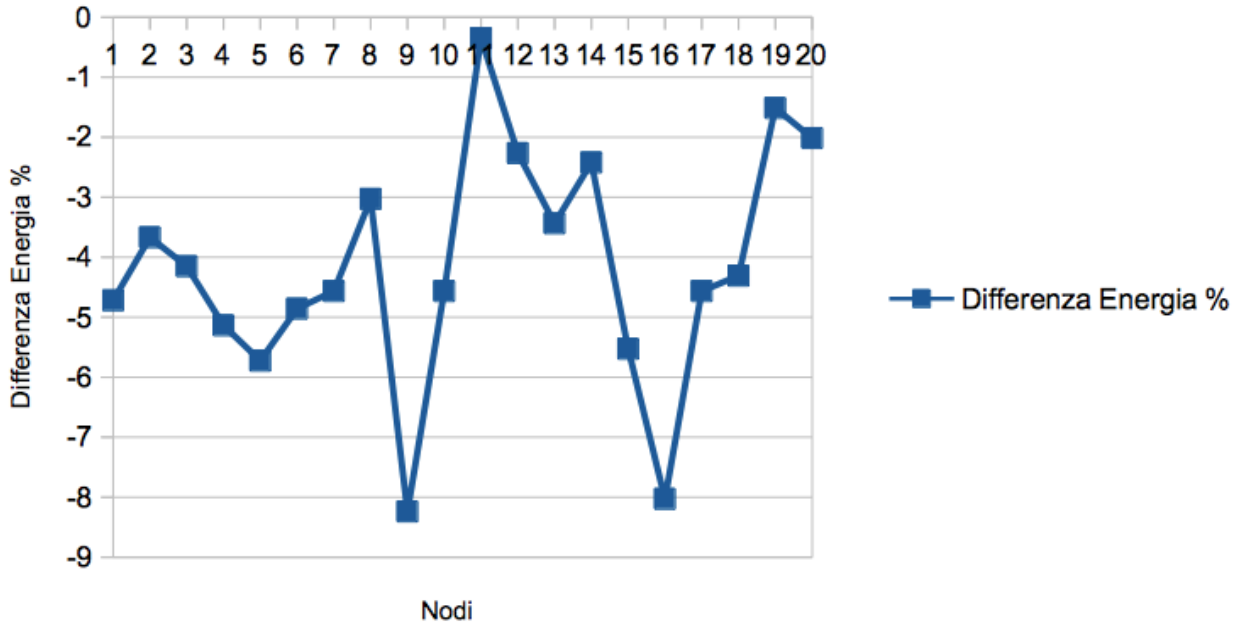
Il confronto che effettuato è stato quello del consumo dell'energia dei vari nodi nei due differenti protocolli (DSR e DSR con DTN), modificando l'intervallo di tempo di rinvio dei pacchetti persi e mantenendo invariate le caratteristiche sopra mostrate.

1.Rinvio pacchetti ogni 100 secondi - Consumo energia al tempo T = 1000 sec
*Rate Trasmissione **5 Pacchetti al secondo***

Nodi	Energia Residua DSR	Energia Residua DTN	Differenza %
N0	54,9529	50,2315	4,7214
N1	56,4477	52,7792	3,6685
N2	52,4477	48,2961	4,1516
N3	58,9198	53,7832	5,1366
N4	50,7398	45,0155	5,7243
N5	57,1709	52,3124	4,8585
N6	53,775	49,2108	4,5642
N7	55,5827	52,5545	3,0282
N8	55,8812	47,6436	8,2376
N9	56,1649	51,6076	4,5573
N10	50,0128	49,6648	0,348
N11	55,3188	53,0537	2,2651
N12	52,1231	48,6873	3,4358
N13	57,348	54,9257	2,4223
N14	51,9106	46,3828	5,5278
N15	56,3248	48,2968	8,028
N16	51,5003	46,9362	4,5641
N17	54,925	50,6172	4,3078
N18	56,8498	55,3363	1,5135
N19	58,1386	56,124	2,0146



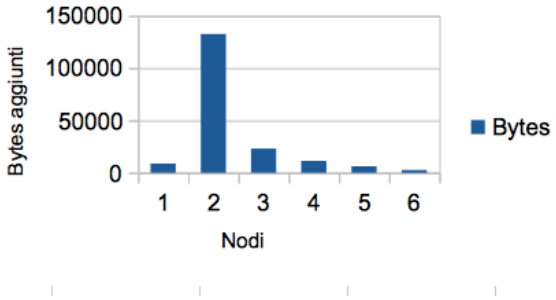
Osservando questo grafico si evince che il consumo di energia della batteria utilizzando l'algoritmo DTN per ogni singolo nodo. Dai dati è possibile capire che l'energia spesa utilizzando l'algoritmo DTN (dovuta al numero maggiore di invio di



pacchetti) è maggiore rispetto a quello DSR con una differenza massima dell' 8%. (come visibile dal grafico successivo).

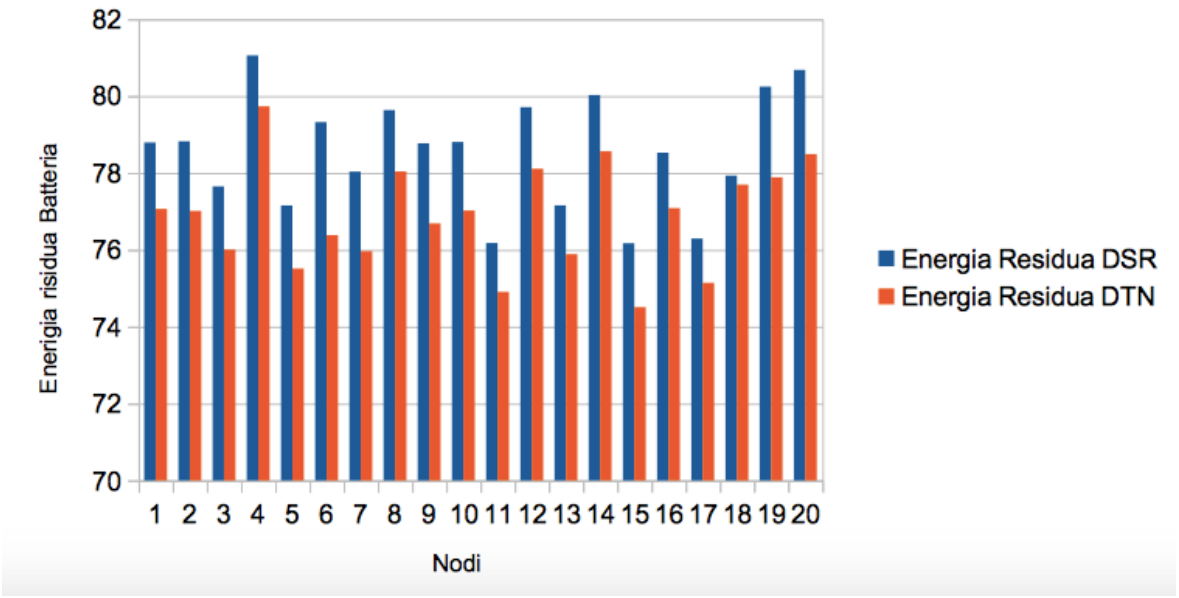
Inoltre è stato calcolato anche il numero totale di pacchetti che vengono memorizzati nella coda che sono pari a 360. Considerando che la dimensione dei pacchetti è 512byte abbiamo un carico di 184.320byte.

	Pacchetti in coda	Bytes
N1	17	8704
N2	259	132608
N3	45	23040
N4	22	11264
N5	12	6144
N6	5	2560
Totale	360	184320

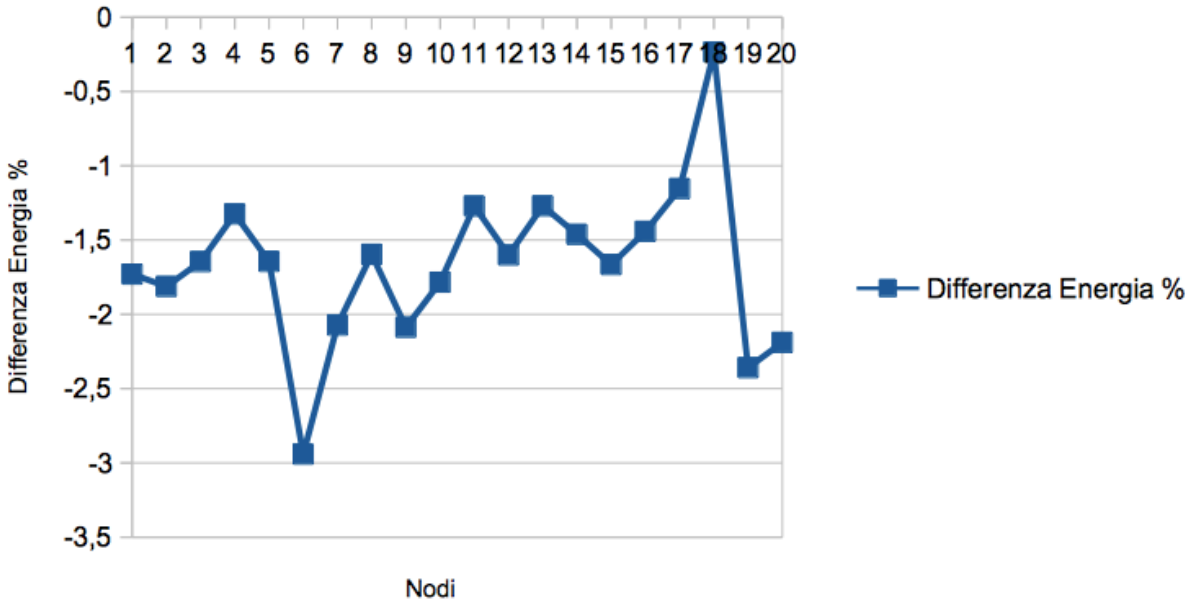


2.Rinvio pacchetti ogni 100 secondi - Consumo energia al tempo T = 1000 sec
Rate Trasmissione ***2 Pacchetti al secondo***

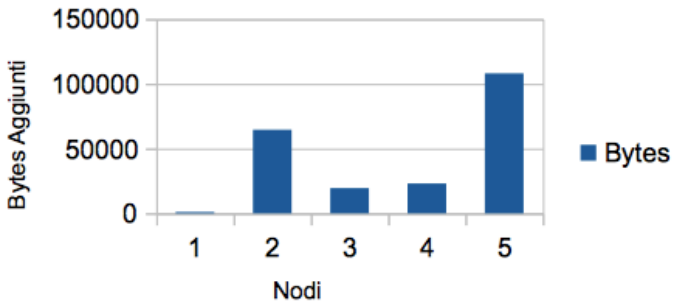
Nodi	Energia Residua DSR	Energia Residua DTN	Differenza Energia %
N0	78,7911	77,0615	-1,7296
N1	78,8221	77,0095	-1,8126
N2	77,6493	76,0041	-1,6452
N3	81,0567	79,732	-1,3247
N4	77,1541	75,5103	-1,6438
N5	79,321	76,3802	-2,9408
N6	78,0336	75,9609	-2,0727
N7	79,6335	78,0373	-1,5962
N8	78,7719	76,6845	-2,0874
N9	78,8041	77,0203	-1,7838
N10	76,1748	74,9032	-1,2716
N11	79,7096	78,1082	-1,6014
N12	77,1544	75,884	-1,2704
N13	80,0253	78,5631	-1,4622
N14	76,1704	74,5047	-1,6657
N15	78,5245	77,083	-1,4415
N16	76,2928	75,1384	-1,1544
N17	77,9293	77,6939	-0,2354
N18	80,2446	77,8853	-2,3593
N19	80,679	78,4898	-2,1892



Anche dall'osservazione questo grafico si evince che il consumo di energia della batteria utilizzando l'algoritmo DTN per ogni singolo nodo, è maggiore rispetto a quello DSR con una differenza massima dell' 3%.(come visibile dal grafico successivo). Ovviamente in questo caso il tasso di invio dei pacchetti è minore, quindi minore è anche il numero di rinvii.



Inoltre è stato calcolato anche il numero totale di pacchetti che vengono memorizzati nella coda che sono pari a 166. Considerando che la dimensione dei pacchetti è 512byte abbiamo un carico di 108032 byte.



	Pacchetti in coda	Bytes
N1	2	1024
N2	126	64512
N3	38	19456
N4	45	23040
Totale	166	108032

Conclusioni

Dopo aver effettuato diverse prove è possibile affermare che utilizzando il modificando il protocollo DSR aggiungendo il protocollo DTN, il vantaggio è sicuramente quello che i pacchetti che precedentemente venivano persi, adesso riescono ad arrivare al destinatario, di contro però il consumo di energia della batteria dei nodi è maggiore, poiché il numero di trasmissioni e ricezioni è maggiore. Per ottimizzare ciò è preferibile aumentare l'intervallo di tempo di ritrasmissione, facendo si che vi siano meno trasmissioni e ricezioni.
Inoltre ogni nodo poiché salva all'interno della struttura dati i pacchetti persi, avrà un carico maggiore.