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.

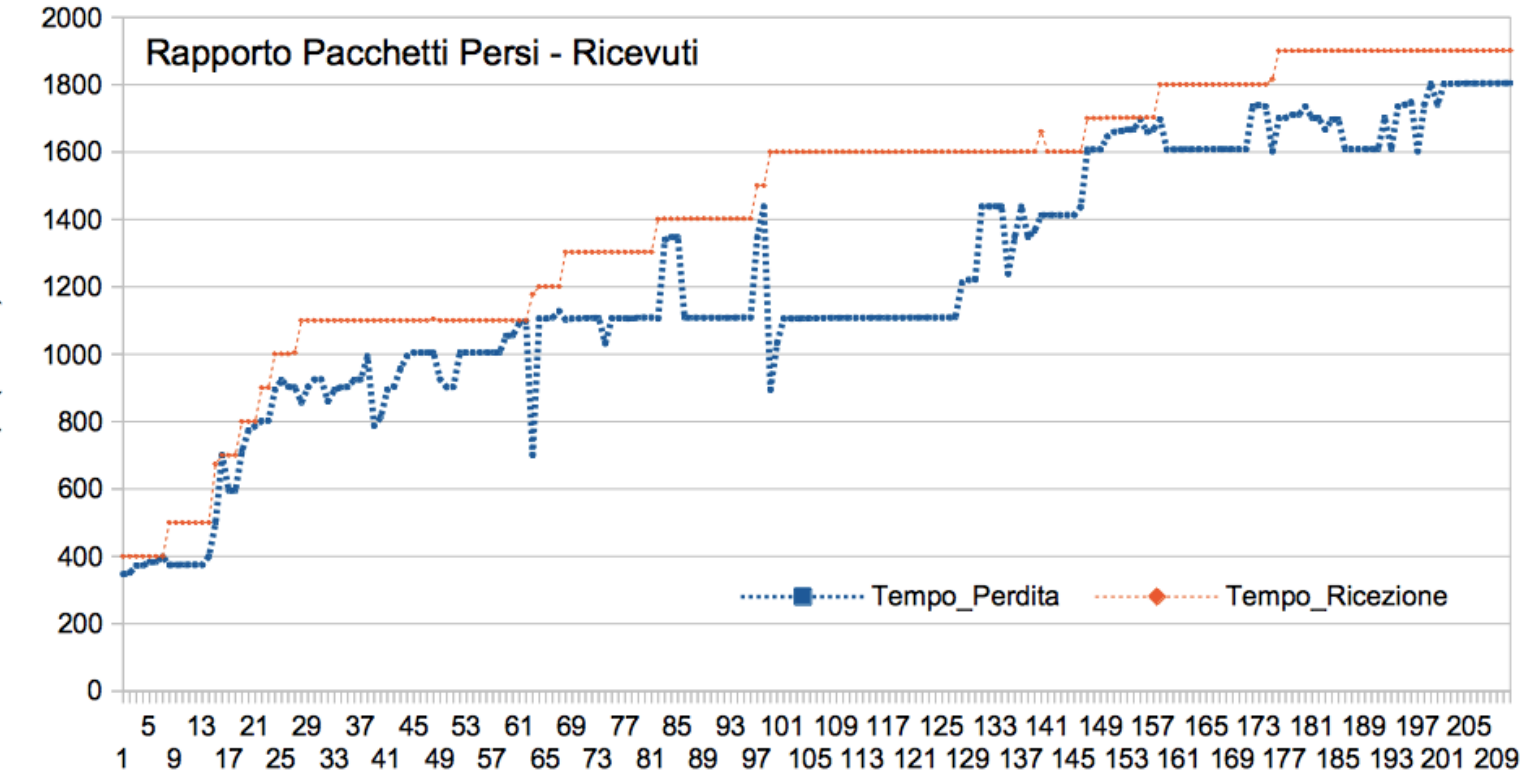
```
if (pacchetti_da_reinviare.size() > 0)
{

  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

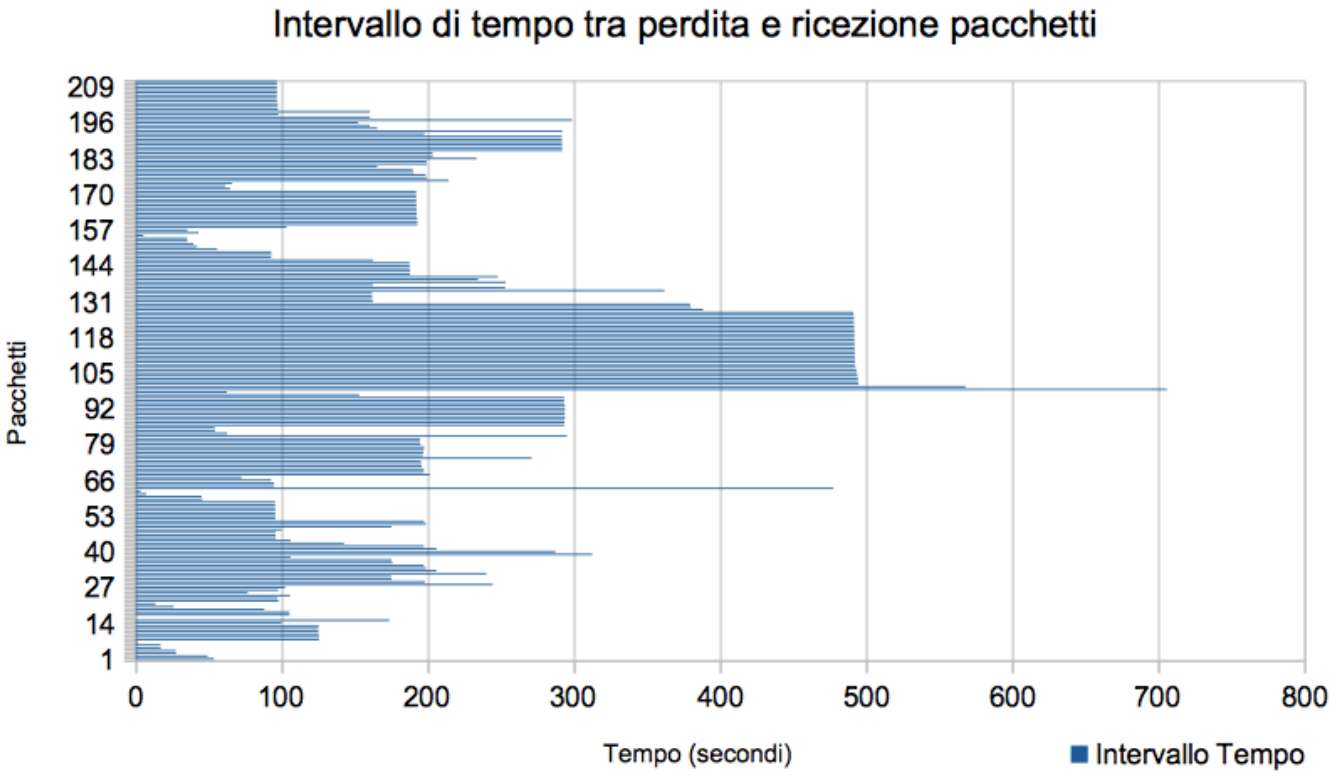
Per constatare le differenze tra l'algoritmo DSR e quello modificato con l'aggiunta del **DTN** è stato creato un ambiente wireless con le seguenti caratteristiche:

1.Rinvio pacchetti ogni 100 secondi - Consumo energia al tempo T = 2000 sec
Rate Trasmissione 0.2 Pacchetti al secondo Traffico cbr - Numero Nodi 10
Numero massimo connessioni 10.

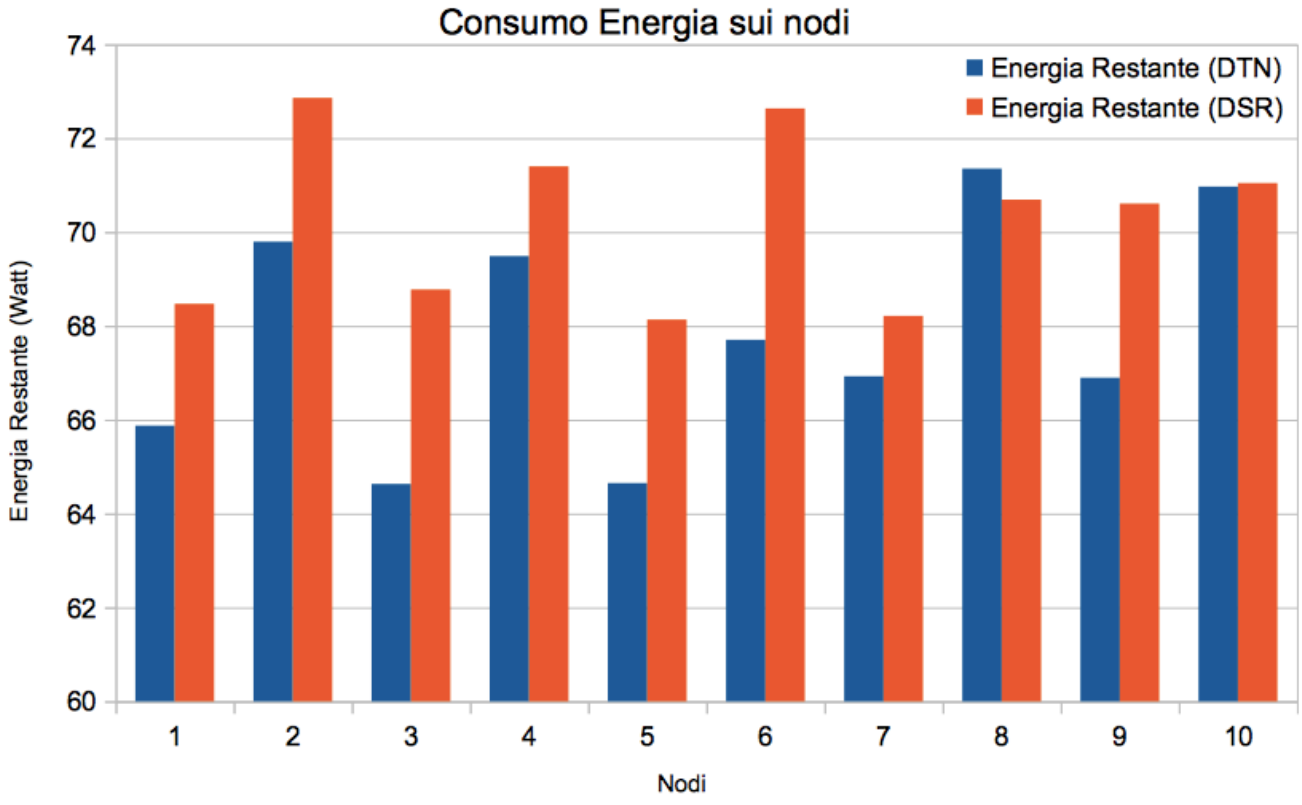
TEMPO SIMULAZIONE 2000 Secondi



Il Grafico mette in relazione Il tempo in cui è il pacchetto è stato perso, con il tempo in cui il pacchetto è arrivato a destinazione a seguito di ritrasmissione. L'intervallo di tempo (asse y) di rinvio dei pacchetti è stato impostato a 100 secondi. Poichè la durata della simulazione è stata settata a 2000 secondi , il numero di intervalli è 10. Per capire meglio questo intervallo di tempo ho creato il seguente grafico.



Effettuando una media degli intervalli di tempo ricavati da questo grafico, il tempo medio di ricezione di un pacchetto dati perso (considerando il tempo di memorizzazione ed il numero di rinvii è pari a **207,77 secondi.**



Un altro aspetto esaminato durante la simulazione è la differenza di Energia restante nei nodi, dopo l'applicazione del protocollo DSR e DTN. La quantità di energia iniziale impostata è stata pari a **100 W**. Il consumo di energia è pari a **0.1 W** per la *ricezione*, **0.3 W** per la fase di *trasmissione* e **0.02 W** durante la fase di *sleep*.

Nodi	Energia Restante (DTN)	Energia Restante (DSR)	Differenza %	Media
N0	65,873	68,469	2,596	**2,453**
N1	69,794	72,8574	3,0634	
N2	64,6302	68,7744	4,1442	
N3	69,4872	71,3974	1,9102	
N4	64,6496	68,1341	3,4845	
N5	67,7026	72,6325	4,9299	
N6	66,9258	68,2128	1,287	
N7	71,3508	70,6886	-0,6622	
N8	66,8951	70,6057	3,7106	
N9	70,9672	71,0413	0,0741	

Effettuando una media è possibile affermare che utilizzando il protocollo DTN vi è un consumo di energia neo singoli nodi superiore al consumo di energia utilizzando il protocollo DSR del **2,453%.** Questo fattore è dovuto al rinvio dei dati, quindi ogni nodo passa più volte dalla fase di trasmissione e ricezione.

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.