UNIVERSITÀ DEGLI STUDI DI PADOVA
Facoltà di Scienze MM. FF. NN.
Corso di Laurea Magistrale in Informatica

Tesi di Laurea Magistrale

Titolo della tesi

Studente:
Daniele Bonaldo

Relatore:
Prof. Claudio Enrico Palazzi

Anno accademico 2010-2011

Indice

1 Introduzione **1**

2 ONE - L'ambiente di simulazione **3**
2.1 Configurazione 3
2.2 Visualizzazione 5
2.3 Reports 7
2.4 Esecuzione 9
2.5 Casualità 10
2.6 Limitazioni 10

3 Modelli di movimento **11**
3.1 Random Map-Based Movement 11
3.2 Shortest Path Map-Based Movement 11
3.3 Routed Map-Based Movement 12
3.4 Working Day Movement Model 12
3.4.1 Esempio di giornata 13
3.4.2 Home Activity Submodel 14
3.4.3 Office Activity Submodel 14
3.4.4 Evening Activity Submodel 15
3.4.5 Transport Activity Submodel 15

4 M2MShare **17**
4.1 Modulo DTN 17
4.1.1 Elezione del Servant 18
4.2 Modulo di ricerca 20
4.3 Modulo di Trasporto 21
4.3.1 Sistema di accodamento 21
4.3.2 Scheduler ed esecuzione di un task 22
4.3.3 Tipi di task 23
4.4 Modulo di Routing 23
4.5 Modulo MAC 23

4.5.1 Presence Collector 23

5 Implementazione **25**
5.1 Modifiche a ONE 25
5.1.1 Gestione File 25
5.1.2 GUI 29
5.1.3 Reports 31
5.2 M2MShare Implementation 34
5.2.1 M2MShareRouter 35
5.2.2 PresenceCollector 36
5.2.3 QueuingCentral 36
5.2.4 Activity 36
5.2.5 Scheduler 37
5.2.6 BroadcastModule 37
5.2.7 IntervalMap 37

6 La simulazione **39**

7 Conclusioni **41**

Bibliografia **43**

Capitolo 1

Introduzione

Una bellissima introduzione

Capitolo 2

ONE - L'ambiente di simulazione

L'ambiente scelto per svolgere le simulazioni è The ONE (Opportunistic Network Environment) versione 1.4.1, descritto in [1]. Questo simulatore scritto in Java è completamente configurabile e permette di simulare il movimento dei vari nodi partecipanti alla simulazione, gestire le connessioni e lo scambio di messaggi fra i vari nodi (utilizzando diversi protocolli di routing) e visualizzare sia i movimenti che il traffico dati nell'interfaccia grafica di cui dispone.

Per quanto riguarda il movimento, il simulatore può accettare in input tracce provenienti da registrazioni dal mondo reale, dati creati tramite generatori di movimento esterni oppure crearli dinamicamente tramite dei modelli di movimento, alcuni dei quali descritti più nel dettaglio in [movimento]. Sia i modelli di movimento che i vari protocolli di routing vengono gestiti come moduli indipendenti e vengono caricati a seconda di quanto impostato in fase di configurazione. Ciò permette una semplice implementazione di nuovi protocolli di routing e modelli di movimento all'interno del simulatore.

Il simulatore permette infine di salvare dati statistici riguardanti le simulazioni svolte tramite la generazione di report, anch'essi gestiti in maniera totalmente modulare e configurabile.

2.1 Configurazione

Una determinata simulazione viene impostata creando dei files di configurazione che descrivano i vari aspetti dello scenario, dalla durata della simulazione al numero di nodi che la compongono, fino alle caratteristiche

specifiche di ogni gruppo di nodi. Tali files di configurazione sono dei semplici files di testo in cui vengono impostati i parametri relativi allo scenario, ai modelli di movimento e ai protocolli di routing. I valori impostati verranno caricati dinamicamente ed andranno ad attivare e configurare i moduli che si è scelto di utilizzare per la simulazione corrente.

All'interno dei files di configurazione, i parametri sono salvati come coppie chiave-valore. La sintassi della maggior parte delle variabili è del tipo

$$Namespace.chiave = valore$$

Il namespace indica generalmente a quale parte dell'ambiente di simulazione la variabile si riferisce. Più nello specifico il namespace indica (nella quasi totalità dei casi) la classe Java che andrà a leggere quel parametro. durante la fase di inizializzazione. Questa convenzione è utilizzata soprattutto dai moduli relativi ai modelli di movimento e ai protocolli di routing, quindi è bene che venga utilizzata nella realizzazione di nuovi moduli da aggiungere al simulatore.

Per facilitare la lettura e la configurazione, i valori numerici possono utilizzare i suffissi kilo (k), mega (M) o giga (G), assieme al punto "."come separatore decimale. I parametri di tipo booleano invece accettano i valori "true"o "1", "false"o "0 ".
Dei commenti possono essere inseriti nei files di configurazione utilizzando il carattere "#", che ottiene il risultato di far saltare il resto della riga durante la fase di lettura.

Per ogni simulazione ci possono essere più files di configurazione, in modo da poter dividere i parametri in più categorie, ad esempio in un file inserire i parametri relativi allo scenario, con le strade e i quartieri in cui è diviso, in un altro file configurare i nodi con le caratteristiche tipiche di ogni gruppo, in un altro ancora selezionare i report da generare durante l'esecuzione e così via. Il primo file di configurazione letto, se esiste, è sempre il file "default_settings.txt"e negli altri files di configurazione è possibile definire nuovi parametri o sovrascrivere alcuni di quelli definiti nei files precedenti. Ciò permette di definire in alcuni files dei parametri generali, validi per tutte le simulazioni, e variarne altri cambiando solo gli ultimi files di configurazione.

All'interno dello stesso scenario, i nodi sono divisi in più gruppi, composti da un numero variabile di nodi. All'interno di un gruppo, tutti i nodi condividono delle caratteristiche comuni, dal protocollo di routing utilizzato, al

modello di movimento, ai tipi di interfacce disponibili per la comunicazione. E' possibile impostare inoltre delle caratteristiche comuni a tutti i gruppi in modo da non dover ripetere la definizione di alcuni parametri per tutti i gruppi. La variazione di caratteristiche fra un gruppo e l'altro permette di ottenere dell'eterogeneità fra i comportamenti simulati all'interno dello scenario.

Nei files di configurazione utilizzati da ONE è possibile inoltre impostare array di valori per ogni parametro: ciò permette, durante una serie di esecuzioni batch, di ottenere automaticamente una serie di simulazioni che differiscono l'una dall'altra per alcune impostazioni di parametri, automatizzando notevolmente la raccolta di dati con configurazioni differenti dello stesso scenario.
In questo caso la sintassi sarà del tipo

Namespace.chiave = [valoreEsecuzione1; valoreEsecuzione2; valoreEsecuzione3; ecc]

Alcuni parametri, infine, accettano come valore il percorso di un file e in questo caso può essere espresso sia in maniera assoluta che relativa. Un esempio di variabili che necessitano di questo tipo di valori sono le mappe, per cui si impostano i files che le contengono, o gli input per i generatori di eventi, per i quali i files da caricare descrivono gli eventi da creare durante la simulazione.

2.2 Visualizzazione

La principale modalità di visualizzazione fornita da ONE è quella tramite la GUI, che permette di seguire in tempo reale l'avanzamento della simulazione. Nella finestra principale è possibile osservare i movimenti dei vari nodi e, selezionandone uno specifico, ottenere informazioni riguardo le connessioni attive, i messaggi trasportati e altri dettagli. E' disponibile inoltre un riquadro in cui viene costantemente aggiornato un log di eventi generati durante la simulazione che possono essere filtrati a seconda di ciò che più interessa (ad esempio visualizzare solo le nuove connessioni o gli scambi di messaggi).

Nel caso di modelli di movimento basati su una mappa, selezionando un nodo sarà possibile vedere il percorso seguito, la destinazione da raggiungere e ottenere informazioni avanzate riguardanti lo stato di quel nodo (connessioni, messaggi trasportati, ecc), come si può vedere in Figura 5.1. La visualizzazione è personalizzabile zoomando, modificando la velocità di avanzamento e

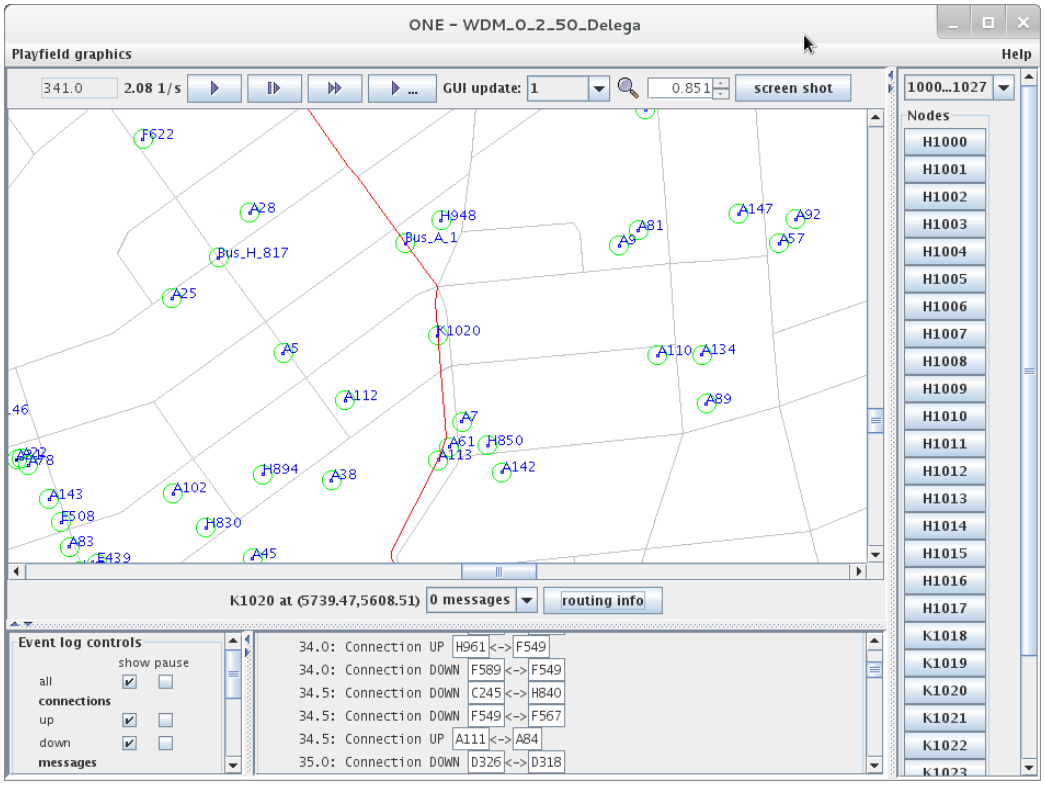

Figura 2.1: Schermata Principale - La schermata principale del simulatore ONE

anche inserendo nello sfondo un'immagine, come ad esempio una carta stradale o una fotografia satellitare della zona interessata.

L'altra modalità di seguire l'avanzamento di una simulazione è la lettura dei reports generati dai vari moduli durante l'esecuzione. Come i modelli di movimento e i protocolli di routing, i generatori di reports sono caricati dinamicamente all'avvio della simulazione, a seconda di ciò che è stato impostato nei files di configurazione. Questa modalità di visualizzazione è particolarmente utile quando non si utilizza la GUI, ma si eseguono più simulazioni in batch, ottenendo quindi alla fine i report con i dati raccolti durante le varie simulazioni.

La modalità batch è indicata nel caso si debbano eseguire più simulazioni in serie senza essere interessati a seguirne l'avanzamento in modalità grafica ma piuttosto valutandone i risultati una volta completate. In questo caso si dimostra molto utile la possibilità di specificare array di valori per i parametri di configurazione, in modo da poter programmare in anticipo le differenze fra le varie simulazioni della serie. Una volta avviato quindi il batch di simulazioni, ONE si occuperà si eseguirle in sequenza e alla massima velocità permessa della macchina in uso e salverà i reports generati in più files impostati durante la configurazione.

2.3 Reports

Il simulatore può gestire la generazione di più reports relativi alla simulazione in esecuzione. Questi report vengono creati da dei moduli attivati in fase di configurazione e consistono generalmente in files di testo in cui vengono salvati i dati e statistiche che verranno poi analizzati a simulazione terminata. Nella versione 1.4.1 di ONE, il sistema permette di generare reports relativi a:

messaggi, che includono numero di messaggi creati, scambiati, scaduti, ecc

contatti, in cui viene indicato il contact e l'inter-contact time fra i vari nodi, oltre al totale dei contatti durante la simulazione.

connessioni, che descrivono l'alternarsi di stato delle connessioni

Come per le altre parti di cui ONE è composto, anche i generatori di reports vengono gestiti come moduli, ed è quindi possibile aggiungerne di nuovi a seconda delle esigenze e dei dati da raccogliere nella singola simulazione.

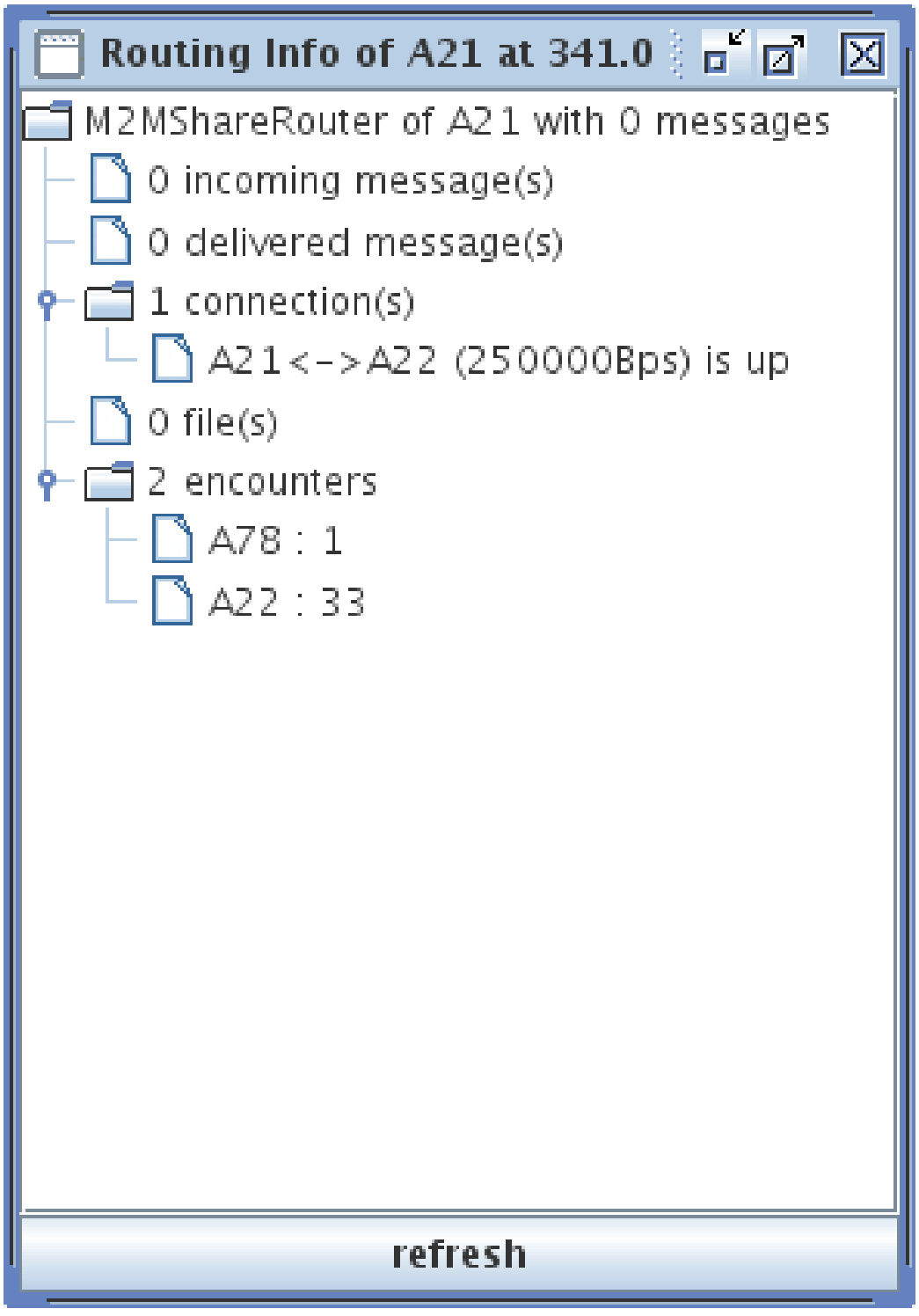

Figura 2.2: Routing Info - Un esempio di finestra in cui sono presenti i dettagli relativi allo stato di un nodo

2.4 Esecuzione

Vale la pena di soffermarsi sull'esecuzione di una singola simulazione, vedendo quindi quali sono i vari passi che vengono eseguiti.

La prima azione svolta dal simulatore è quella di caricare le impostazioni dai vari files di configurazione, la cui posizione viene passata per parametro al momento dell'esecuzione del simulatore. Man mano che un nuovo file di configurazione viene letto, i valori dei parametri in esso contenuto vanno ad impostare il valore di alcune variabili dell'ambiente di simulazione, sovrascrivendone anche il valore nel caso fossero già state impostate.

Una volta caricate le impostazioni relative alla simulazione, viene creato lo Scenario. Questo contiene al suo interno tutti gli elementi attivi durante la simulazione (come i nodi, i generatori di reports e quelli di messaggi), così come quelli passivi (ad esempio le mappe che compongono il mondo simulato). In questa fase vengono quindi creati tutti i nodi partecipanti alla simulazione, ognuno dotato di un proprio modello di movimento, un router configurato secondo le caratteristiche del gruppo di nodi e una serie di *listener* per la cattura di eventi e la successiva generazione di reports.

Quando tutte le entità necessarie all'esecuzione sono stati create e configurate, si passa all'esecuzione vera e propria. Questa consiste nel ripetere l'aggiornamento dello stato del mondo, chiamando un metodo *update()*, e incrementare il valore del tempo simulato fino al raggiungimento di un tempo impostato come fine simulazione. L'incremento temporale che viene effettuato ad ogni aggiornamento è impostato nei files di configurazione (con il parametro *Scenario.updateInterval*), espresso in secondi, ed influenza i vari modelli di movimento dei nodi. La prima operazione svolta durante l'*update()* del mondo simulato è lo spostamento dei vari nodi, che avviene a seconda del modello di movimento adottato dal nodo e dell'incremento temporale applicato. Ad esempio un nodo che simula un automobilista si sposterà maggiormente rispetto ad uno che simula un pedone, a parità di intervallo di tempo simulato. Come vedremo nella sezione 3 relativa ai modelli di movimento, questi possono essere anche molto complessi e simulare diversi comportamenti a seconda delle configurazioni adottate.

Una volta effettuato il movimento, per ogni nodo viene aggiornato lo stato delle connessioni e del router simulato. Per ogni interfaccia di rete disponibile viene quindi aggiornato lo stato a seconda che lo spostamento abbia comportato una caduta della connessione o abbia permesso di entrare

nel raggio di comunicazione di un interfaccia di rete relativa ad un altro nodo. Ogni qualvolta lo stato di una connessione cambia, vengono avvisati i *listeners* interessati, per la generazione di report, e viene aggiornata la visualizzazione grafica della connessione, se attiva, come si può vedere, ad esempio, in Figura 2.3.

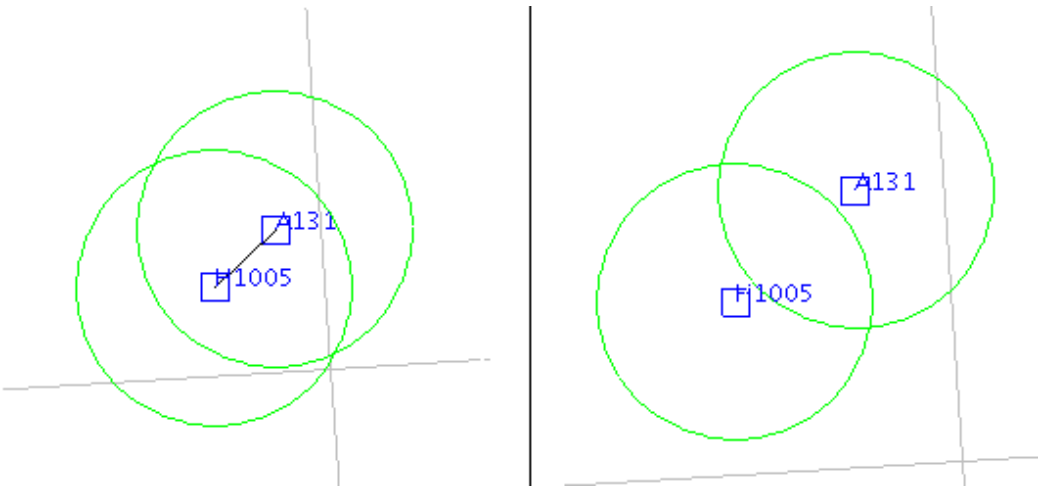

Figura 2.3: connessioni - Un esempio di connessioni tra nodi tratto dalla finestra principale di ONE. Nella parte sinistra dell'immagine si può notare la connessione attiva, mentre nella parte a destra i due nodi non sono più l'uno nel raggio di comunicazione dell'altro, indicato dal cerchio verde attorno al nodo.

L'ultima parte dell'aggiornamento relativo allo stato di un nodo riguarda l'aggiornamento del router. Questo è fortemente dipendente dal protocollo di routing implementato ed è proprio nell'esecuzione del metodo *update()* relativo al router che si svolgono le azioni caratteristiche di un protocollo rispetto ad un altro.

2.5 Casualità

come viene introdotta della casualità nelle varie simulazioni

2.6 Limitazioni

non più in basso del livello di routing simulazione temporale discreta mancanza di simulazione di un file system

Capitolo 3

Modelli di movimento

Uno degli aspetti più importante per garantire la realisticità di una simulazione riguarda il comportamento dei nodi, in particolare il loro movimento. Come è facile intuire, in una DTN i cambiamenti di posizione dei vari nodi influiscono pesantemente la connettività e quindi le prestazioni dell'intera rete. Proprio per questo motivo è fondamentale simulare il più realisticamente possibile il comportamento dei nodi all'interno della simulazione.

Il simulatore TheONE contiene al suo interno diversi modelli di movimento che possono essere assegnati ai vari gruppi di nodi. Presenterò ora un breve riassunto dei modelli a disposizione, soffermandomi in particolare su quello scelto per le simulazioni svolte, ossia il Working Day Movement Model (WDM)

3.1 Random Map-Based Movement

Il modello più semplice fra quelli disponibili è il Random Map-Based Movement (MBM). I nodi che adottano questo modello si muovono in maniera casuale ma sempre seguendo le strade descritte dalla mappa della simulazione. Il risultato di questa modalità è un movimento troppo casuale per simulare accuratamente il comportamento umano per quanto riguarda la mobilità.

3.2 Shortest Path Map-Based Movement

Lo Shortest Path Map-Based Movement (SPMBM) è un modello leggermente più realistico rispetto a MBM. In questo modello i nodi scelgono casualmente un punto di destinazione all'interno della mappa e seguono la via

più breve per raggiungerlo dalla posizione attuale, sempre seguendo le strade descritte dalla mappa della simulazione. Le destinazioni possono essere scelte in maniera completamente casuale o da un insieme di Punti di Interesse, in modo da simulare l'attrazione delle persone per luoghi come punti di ristoro, negozi o attrazioni turistiche.

3.3 Routed Map-Based Movement

Un modello che invece della casualità nel movimento utilizza dei percorsi predeterminati è il Routed Map-Based Movement (RMBM). I nodi che adottano questo modello si muovono lungo rotte predefinite, per tutta la durata della simulazione, rendendo RMBM utile per rappresentare degli spostamenti ripetitivi, come ad esempio quelli di autobus, tram o treni.

3.4 Working Day Movement Model

I modelli finora descritti sono senz'altro di semplice comprensione e realizzazione, oltre ad essere molto efficienti per quanto riguarda le prestazioni, ma non forniscono una realistica rappresentazione del movimento umano, soprattutto per quanto riguarda i valori di inter-contact e contact time.
Un modello che genera dei valori più realistici per questi parametri, rappresentando più realisticamente quindi il movimento umano, è il Working Day Movement Model (WDM). Come il nome può fare intuire, questo modello simula gli spostamenti compiuti da una persona durante una tipica giornata lavorativa e in [2], dove è descritto il modello, è evidenziato anche come i valori generati seguano realisticamente quelli trovati utilizzando dati di spostamento provenienti da tracce reali.
Una giornata simulata comprende le seguenti attività principali svolte dai vari nodi:

- dormire a casa
- lavorare in ufficio
- uscire alla sera con gli amici

Ovviamente le attività potrebbero cambiare enormemente a seconda dello stile di vita e del lavoro svolto dalle singole persone, ma queste tre attività sono le più comuni e possono essere associate alla tipica giornata di una gran quantità di persone. La ripetitività giornaliera delle azioni e il fatto di svolgerle in luoghi comuni a più persone permette la formazione spontanea

di comunità: persone che vivono e dormono nella stessa casa formeranno una famiglia, più persone che lavorano nello stesso ufficio agli stessi orari saranno colleghi di lavoro, mentre degli amici si possono trovare ad orari comuni alla sera per uscire assieme. La creazione di queste comunità non viene mostrata da modelli più semplici quali RMBM o SPMBM.

Per la simulazione delle attività giornaliere, WDM utilizza dei sottomodelli dedicati, oltre a dei sottomodelli preposti a simulare gli spostamenti fra un'attività e l'altra. Una persona si potrà quindi spostare a piedi, in auto o utilizzando i mezzi pubblici, a seconda della propria disponibilità e convenienza. Il fatto di muoversi da soli o in gruppo (prendendo lo stesso bus o camminando assieme la sera) permette di avere dei comportamenti eterogenei e quindi migliorare ulteriormente la realisticità degli spostamenti compiuti dai vari nodi.

3.4.1 Esempio di giornata

Durante una tipica giornata il punto di partenza per ogni nodo è la propria abitazione. Ogni nodo ha un orario di sveglia assegnato, generato utilizzando una distribuzione normale con media pari a 0 e deviazione standard impostabile in fase di configurazione, che indica l'orario in cui la persona uscirà di casa. Il valore viene generato per ogni ad inizio simulazione, rimarrà lo stesso per tutti i giorni successivi e la differenza fra i valori di vari nodi sta ad indicare la differenza fra i diversi stili di vita nella vita reale (ad esempio una persona che impiega pochi minuti a prepararsi la mattina rispetto a chi impiega ore anche solo per fare colazione).

Una volta usciti di casa, i vari nodi si dirigono al lavoro utilizzando l'auto (se disponibile) o a piedi oppure utilizzando i mezzi pubblici, a seconda di quale sia il metodo più conveniente. Conseguentemente alla scelta del mezzo di trasporto viene utilizzato il corrispondente sottomodello.

Una volta raggiunto il luogo di lavoro, la persona ci resta per la durata della sua giornata lavorativa e quindi decide, con una determinata probabilità, se tornare direttamente a casa o spostarsi per un'attività serale. Anche in questo caso gli spostamenti vengono gestiti utilizzando i corrispondenti sottomodelli.

3.4.2 Home Activity Submodel

Ogni nodo ha una posizione impostata come Home Location, che viene utilizzata come punto di partenza alla mattina e punto di ritorno alla sera: una volta tornata a casa una persona si muove per una breve distanza e poi resta ferma fino all'orario di risveglio, la mattina successiva. Questo comportamento non è un errore, ma simula il fatto di lasciare il telefono su di un tavolo o in carica fino al momento di uscire nuovamente di casa, mentre la persona svolge le normali attività domestiche come mangiare, guardare la TV o dormire.

3.4.3 Office Activity Submodel

Il sottomodello relativo all'attività lavorativa è un modello bidimensionale che simula il comportamento di una persona all'interno di un ufficio, in cui è posizionata la propria scrivania e dalla quale ogni tanto si alza per partecipare ad una riunione, parlare con un collega o, perché no, per una pausa caffè. Durante tutti questi momenti, come è facile intuire, è possibile che il nodo entri in contatto con nodi relativi ad altri colleghi di lavoro.

L'ufficio è descritto come un'unica stanza con pianta rettangolare, in cui l'unico punto di ingresso, la porta, è l'angolo in alto a sinistra e ogni persona che vi lavora ha una scrivania posizionata in un determinato punto. Non viene simulata la presenza di muri all'interno della stanza, che quindi verrà descritta come un luogo più grande del normale, in modo da simulare il tempo impiegato per superare ostacoli, nel movimento dalla scrivania ad una destinazione interna all'ufficio.

Una volta entrato, l'impiegato si muove subito camminando verso la propria scrivania, dove rimane per un periodo di tempo casuale, generato utilizzando una distribuzione di Pareto. Passato questo tempo il nodo sceglie una destinazione casuale all'interno dell'ufficio, cammina fino a raggiungerla e quindi attende per un periodo di tempo casuale generato utilizzando la stessa distribuzione di Pareto prima di tornare alla propria scrivania. La ripetizione del movimento dalla scrivania ad una posizione casuale interna all'ufficio continua fino al termine della giornata lavorativa, che ha una durata impostabile in fase di configurazione.

I parametri della distribuzione possono essere impostati per ogni gruppo di nodi, in modo da simulare diversi tipi di attività all'interno del luogo di lavoro, dall'insegnante che ogni ora si deve spostare in un'aula diversa,

ad un commesso che non lascia mai la propria postazione per tutto l'orario lavorativo.

3.4.4 Evening Activity Submodel

Il sottomodello Evening Activity simula attività che possono essere svolte dopo lavoro, nel tardo pomeriggio - sera, come andare a fare shopping, in un bar o a mangiare in una pizzeria o ristorante. Tali attività vengono svolte in gruppo e con una probabilità configurabile, che determina se la persona torna o meno subito a casa dopo il lavoro.

Al termine della giornata lavorativa, il nodo si sposta verso il proprio luogo d'incontro preferito, che è una posizione impostata all'inizio della simulazione. Una volta arrivato attende che lo raggiungano un numero di persone sufficientemente elevato per formare un gruppo e cominciare quindi l'attività. Il numero massimo e minimo di persone che possono formare un gruppo è configurabile e quando tutti i gruppi per un determinato punto di ritrovo sono al completo ne viene creato un altro.

Una volta che tutti i componenti del gruppo legato all'attività sono arrivati, camminano assieme per una breve distanza verso una destinazione scelta casualmente e quindi si fermano per un tempo più lungo, generato casualmente all'interno di valori preimpostati. Una volta terminata questa pausa (finita la cena, lo shopping o la visione di un film al cinema), le varie persone si separano e tornano verso casa.

3.4.5 Transport Activity Submodel

Il Transport Activity Submodel è il sottomodello incaricato di gestire gli spostamenti dei nodi fra le diverse attività.

All'inizio della simulazione ad ogni nodo viene assegnata un'auto con una probabilità configurabile. Le persone che la possiedono la utilizzeranno per tutti gli spostamenti, mentre chi ne è sprovvisto si muoverà a piedi o utilizzando un mezzo pubblico. L'eterogeneità di mezzi di trasporto utilizzati permette di simulare realisticamente i movimenti di diverse tipologie di persone ed inoltre ha impatto anche sul protocollo di routing utilizzato, in quanto nodi che si muovono utilizzando mezzi propri si sposteranno più velocemente, permettendo così il trasporto più rapido di pacchetti per lunghe distanze.

A seconda del mezzo di trasporto utilizzato, quindi, il Transport Activity Submodel si rifà a tre sottomodelli distinti:

Walking Submodel : i nodi che non possiedono un'auto si muovono camminando lungo le strade ad una velocità costante, utilizzando l'algoritmo di Dijkstra per provare il percorso più breve dalla posizione corrente alla destinazione.

Car Submodel : i nodi che possiedono un'auto muovono più velocemente dei pedoni, durante le transizioni fra attività, ma si muovono come gli altri nodi all'interno di una singola attività. Non vengono considerati traffico e semafori durante la guida e ogni auto può portare una sola persona (il car sharing non ha ancora avuto successo nel mondo simulato).

Bus Submodel : nella città possono essere presenti più linee di trasporti pubblici (tram, autobus, funivie), ognuna delle quali viene percorsa da più mezzi ad orari predefiniti. Ogni mezzo pubblico può trasportare più persone.

Ogni persona che non possiede un'auto conosce una linea di mezzi pubblici e può utilizzare qualunque mezzo appartenente a quella linea. Il fatto di prendere il mezzo pubblico o di camminare dipende da un confronto di distanze euclidee: la distanza fra il luogo di partenza e quello di destinazione oppure quella fra il luogo di partenza e la fermata più vicina sommata alla distanza fra la destinazione e la fermata più vicina alla destinazione. Nel caso sia minore la prima allora il nodo camminerà fino alla destinazione, altrimenti utilizzerà i mezzi pubblici. Per fare ciò camminerà fino alla fermata più vicina, utilizzando il Walking Submodel, attenderà il primo mezzo che passerà per quella linea nella direzione corretta, utilizzerà il Bus Submodel fino alla fermata più vicina alla destinazione e quindi tornerà ad utilizzare il Walking Submodel camminando fino al punto di arrivo.

Capitolo 4

M2MShare

La parte principale: si parla del protocollo Differenza di operare in una DTN rispetto ad una rete tradizionale importanza del routing e delle deleghe

4.1 Modulo DTN

Le DTN sono particolarmente utili nel caso di situazioni in cui la connettività non è costante e possono esserci degli intervalli di tempo lunghi e soprattutto imprevedibili, fra un contatto e l'altro. Un esempio evidente è quello dell'ambito mobile, in cui l'utilizzo di DTN si presta particolarmente per il superamento degli ostacoli sopraccitati.

I protocolli di routing tradizionalmente utilizzati per reti ad-hoc non sono adatti all'utilizzo su reti mobili in cui ogni nodo dispone di una connessione con portata limitata, in quanto spesso non c'è la possibilità di instaurare una connessione end-to-end fra due nodi comunicanti. Per un'applicazione di file sharing, inoltre, è fondamentale un'elevata disponibilità di file è fondamentale per garantire il recupero di un file cercato, e questa disponibilità si traduce nella necessità di un elevato numero di nodi connessi fra loro, cosa che come abbiamo visto non è affatto scontata in una rete mobile.

Per ovviare a questo M2MShare utilizza un metodo di comunicazione asincrono i cui un peer *client* (alla ricerca di un file), può delegare ad un altro peer *servant* la ricerca e il successivo recupero di un file. Il principio delle deleghe è quindi alla base di tutto il sistema di M2MShare, in quanto permette di ampliare enormemente il raggio d'azione di una ricerca, in una

rete per definizione non connessa e formata da nodi sparsi.

E' utile approfondire il concetto di delega: questa consiste nella richiesta da parte di un client verso un server dell'esecuzione di un determinato task; il tipo del task può variare da una query composta da più keywords, per la quale si vuole ricevere una serie di files che soddisfino la query, alla ricerca di un determinato file fra i vari nodi della rete, il cui obiettivo è ottenere l'intero file o parti del file cercato. Quando un servant riceve una delega, la schedula per eseguirla in seguito e cercare ciò che viene richiesto. Una volta che una delega ricevuta è stata soddisfatta, il servant crea un nuovo task di tipo forward, il cui scopo è ritornare il risultato della delega al client che l'ha delegata.
In questo modo la mobilità dei nodi viene vista come un aspetto positivo della rete, in quanto permette di entrare in contatto, tramite le deleghe, con nodi che altrimenti un peer client potrebbe non incontrare mai.
Tutti i task delegati hanno poi un TTL entro cui il risultato deve essere ritornato al client. Se un task simile non viene completato, e il suo risultato ritornato, entro lo scadere del TTL, la delega viene considerata come scaduta e quindi non più schedulata per l'esecuzione nel servant.

4.1.1 Elezione del Servant

Visto che tutto il ciclo di vita di un task, dalla creazione alla delega al ritorno del risultato, si svolge in un ambiente infrastructure-less, in cui il percorso dei dati viene creato dinamicamente durante l'esecuzione, è fondamentale una scelta oculata riguardo a chi delegare un proprio task insoddisfatto. Delegare un task a tutti i nodi incontrati, infatti, non è la scelta più conveniente, visto che si tradurrebbe in un prevedibile spreco di traffico oltre che di energia, fattore da non sottovalutare in un ambiente mobile in cui i nodi hanno un autonomia non certo infinita.
Un possibile fattore da valutare potrebbero essere gli interessi in comune: come analizzato in [8], riguardo alla diffusione di dati, un nodo potrebbe diffondere dati differenti a seconda del nodo che incontra. Se è un amico, allora potrebbe essere conveniente diffondere i dati relativi ai loro interessi comuni, mentre se è uno sconosciuto allora i dati diffusi potrebbero essere i più lontani dai loro interessi comuni.

Una volta delegato il task, poi è indispensabile che il risultato del task venga ritornato al client, una volta completato, altrimenti tutto il lavoro svolto è stato inutile. L'idea di base utilizzata in M2MShare è quindi quella

di eleggere come servant dei nodi che ci si aspetta di incontrare di nuovo. Come visto anche nella sezione 3.4 relativa al modello di movimento WDM, le attività quotidiane di una persona, possono generalmente lasciare trasparire una routine di fondo rispetto agli spostamenti e alle persone incontrate. Si vengono spontaneamente a creare delle *comunità* temporanee all'interno delle quali le persone si incontrano con regolarità: ad esempio dei colleghi entrano in contatto ogni giorno in ufficio, dei pendolari utilizzano gli stessi mezzi pubblici per spostarsi alla mattina verso il luogo di lavoro e alla sera per tornare a casa, ecc..

Una persona adatta ad essere eletta come servant è quindi una persona incontrata frequentemente. Per valutare quindi quanto spesso si entra in contatto con un altro device, M2MShare utilizza un demone chiamato *PresenceCollector*, descritto nella sezione 4.5.1. Questo demone è incaricato di tenere traccia del numero di volte che il client è entrato in contatto con ogni altro nodo. Per fare ciò, effettua delle scansioni ad intervalli regolari, tenendo traccia dei nodi presenti all'interno del raggio di comunicazione. Una volta che il numero di incontri supera un parametro detto *Frequency Threshold*, il nodo viene eletto come servant e il client può delegarvi dei task.

La frequenza delle scansioni del *PresenceCollector* è un parametro importante, in quanto influenza sia la probabilità di un nodo di essere eletto come servant (con scansioni più frequenti il numero di incontri per nodo cresce più rapidamente rispetto all'utilizzare una frequenza di scansione minore), sia il consumo energetico del client (scansioni più frequenti implicano un consumo maggiore). In M2MShare la frequenza di scan del *PresenceCollector* è un parametro regolabile dall'utente, mentre uno sforzo aggiuntivo è stato fatto nel regolare automaticamente il valore della *Frequency Threshold*.

Questa è infatti l'altro parametro che influenza pesantemente le probabilità per un nodo di venire eletto come servant: con una *Frequency Threshold* bassa, bastano pochi incontri affinché un nodo la superi e venga quindi eletto, rischiando di delegare un task ad un servant che verrà rincontrato con scarsa probabilità. Con una *Frequency Threshold* troppo elevata invece è possibile che molti nodi promettenti vengano scartati.

M2MShare utilizza quindi un algoritmo di tuning che, all'inizio di ogni giorno, regola il valore della *Frequency Threshold* in base a quanto osservato nel giorno precedente, rispetto al numero di servants eletti e ad un altro parametro, detto *Probation Window*. Affinché un nodo venga considerato periodico, il numero degli incontri con esso, in un periodo di tempo pari al

valore della *Probation Window*, deve essere superiore alla *Frequency Threshold*. Quindi se il valore della *Probation Window* diventa troppo elevato (pochi nodi vengono eletti), si procede a ridurre la *Frequency Threshold*, rendendo così più probabile l'elezione di nodo a servant dopo un minor numero di incontri. Viceversa se il numero di servants attivi è superiore alle previsioni si provvede a diminuire il valore della *Probation Window*. Quando anche questo raggiunge il minimo (impostato a 2 giorni), si procede ad incrementare la *Frequency Threshold*.

La lista contenente il numero di incontri per nodo, utilizza poi una tecnica di rimpiazzo in modo da garantire un consumo contenuto di memoria, garantendo allo stesso tempo agli altri nodi una possibilità di venire eletti come servants. Il criterio utilizzato per decidere se un nodo deve restare o meno nella lista è quello di valutare da quanto tempo vi è inserito: se un nodo viene eletto come servant, allora viene tolto dalla lista in quanto divenuto attivo, ma se un nodo è presente nella lista da un periodo di tempo superiore al valore della *Probation Window*, allora anche in questo caso viene tolto, per dare la possibilità ad altri nodi di essere monitorati e magari venire eletti. Infine, nel caso la lista sia piena ogni nuovo nodo incontrato viene scartato, finché non si libera un posto grazie all'elezione di un nuovo servant o a causa della politica di rimpiazzo descritta.

4.2 Modulo di ricerca

Prima di poter cominciare il recupero di un file interessante per l'utente, è fondamentale che il sistema sappia che file cercare, fra quelli disponibili nella rete. Il modulo incaricato di assolvere questo compito è modulo di ricerca.

Ogni device mantiene un repository in cui sono contenute informazioni relative ai file condivisi con gli altri utenti che utilizzano M2MShare. Queste informazioni includono nome del file, dimensione, posizione nel file system e un hash che lo identifica unicamente nella rete. Questo ultimo valore è particolarmente utile quando la ricerca ha come oggetto un file specifico, rispetto ad una serie di files, permettendo quindi una efficiente query con una risposta di tipo booleano (file presente non presente nel repository).
Oltre alla ricerca orientata al singolo file, il modulo di ricerca permette anche quella tramite l'uso di keywords specificate dall'utente. Per permettere ciò viene utilizzata anche una strategia di indicizzazione comune nell'Information Retrieval, ossia quella dell'*Inverted Index*: ogni file è indicizzato sotto un

certo numero di termini contenuti nella sua descrizione e durante la ricerca è fra questi termini che il modulo andrà a cercare nel caso di richiesta.

4.3 Modulo di Trasporto

Il modulo di trasporto è probabilmente quello più complesso fra i moduli che compongono M2MShare. I suoi compiti spaziano dal gestire il ciclo di vita di un task (dalla creazione al termine passando per la delega), gestire la memoria utilizzata, fino alla gestione delle code di attività e la loro esecuzione.

4.3.1 Sistema di accodamento

Durante l'esecuzione di M2MShare, un nodo può trovarsi a dover eseguire numerose attività derivanti da deleghe ricevute da altri nodi e da task create dall'utente utilizzatore del device stesso. Idealmente ogni attività dovrebbe essere eseguita in un thread dedicato, operando così in parallelo alle altre attività. Questo non è praticamente conveniente, in quanto comporterebbe un utilizzo troppo elevato di memoria, per dei dispositivi come quelli mobili in cui le risorse a disposizione sono limitate.

Per risolvere questo problema M2MShare utilizza un sistema di accodamento in cui, non appena un task viene creato (dal nodo stesso o in seguito a delega), viene subito accodato in una delle code di attività disponibili e schedulato per la successiva esecuzione. In questo modo le varie attività vengono eseguite in maniera sequenziale, a seconda del tipo. Non tutte le attività vengono gestite infatti con la stessa priorità e diverse strategie vengono adottate a seconda della coda in cui un task viene accodato.

Il componente dedicato alla gestione delle code di attività è il *QueuingCentral*, che contiene al suo interno le seguenti code, ognuna delle quali contiene al suo interno attività con caratteristiche diverse:

dtnDownloadQueue: contiene al suo interno task remoti indicanti che un servant ha completato un task delegato in precedenza.

dtnPendingQueue: contiene al suo interno task delegati da altri verso il nodo corrente, che per questi funge da servant.

dtnPendingUpload: contiene i task delegati da altri, che il nodo corrente ha terminato e di cui è pronto a ritornare il risultato.

queryQueue: contiene le query eseguite dall'utente del nodo per la ricerca di files.

uploadQueue: contiene le richieste di dati verso altri nodi.

virtualFileQueue: contiene le richieste di download di file inserite dall'utente del nodo.

Il *QueuingCentral* è anche responsabile di definire delle policy riguardanti la memoria da assegnare alla memorizzazione dei task nelle code, definendo quindi un limite alle deleghe ricevute e al numero massimo di trasferimenti da effettuare.

4.3.2 Scheduler ed esecuzione di un task

Lo *Scheduler* è il componente incaricato di schedulare, come suggerisce il nome, l'esecuzione delle varie attività contenute nelle code del *QueuingCentral*. I vari task vengono smistati fra quattro *execution flows*:

triggered_activity_flow, che interessa i task contenuti nella *dtnPendingUpload*. Viene eseguito ogni volta che il *PresenceCollector* effettua una scansione dei dispositivi presenti nelle vicinanze, in modo da poter eventualmente ritornare il risultato di un task delegato al nodo corrente.

local_activity_flow, che interessa i task contenuti nelle code *dtnDownloadQueue*, *virtualFileQueue* e *queryQueue*. Viene data priorità ai task contenuti nella *dtnDownloadQueue*, in quanto rappresentano dei task delegati per i quali i rispettivi servants sono pronti a ritornare il risultato.

pending_activity_flow, che interessa le attività contenute nella *dtnPendingQueue*, ossia i task delegati da altri al nodo corrente.

upload_activity_flow, che interessa le attività contenute nella *uploadQueue*.

Una volta che un task pronto per essere eseguito viene estratto dalla coda in cui si trova, la sua esecuzione è gestita da un'entità detta *Executor*. Questa è un demone attivo presente nel sistema, che è incaricato fra l'altro di gestire anche il numero di esecuzioni parallele di un singolo task: può infatti capitare, per alcuni tipi di attività come i VirtualFile, che l'esecuzione consista nel download simultaneo di parti di file da diverse fonti presenti nel raggio di comunicazione. In questo caso ogni trasferimento è gestito da una distinta entità detta *Communicator*, entità presenti in numero limitato all'interno di ogni nodo.

4.3.3 Tipi di task

vale la pena descriverli?

4.4 Modulo di Routing

4.5 Modulo MAC

4.5.1 Presence Collector

Capitolo 5

Implementazione

In questa sezione verrà descritta l'implementazione del protocollo realizzata utilizzando come base il simulatore ONE (descritto nel Capitolo 2). Come prima cosa è però necessario soffermarsi su alcune modifiche che àš stato necessario eseguire sul simulatore stesso, in modo da poter poi simulare appieno il comportamento di M2MShare.

5.1 Modifiche a ONE

Come già ampiamente descritto, M2MShare è un protocollo di peer-to-peer, adibito alla condivisione di file fra dispositivi mobili. ONE è allo stato dell'arte per quanto riguarda la simulazione del movimento e delle connessioni fra dispositivi mobili (come analizzato in [11]), ma abbiamo visto che possiede comunque delle limitazioni, le principali elencate nella sezione 2.6. Prima di poter realizzare un'implementazione completa di M2MShare è stato quindi necessario effettuare delle modifiche a ONE, introducendo alcune caratteristiche indispensabili a simulare il comportamento di un protocollo di peer-to-peer per lo scambio di files.

5.1.1 Gestione File

Il primo passo è stato quindi quello di dotare i vari nodi che compongono la simulazione, descritti dalla classe *DTNHost*, di un file system al cui interno salvare i files oggetto delle ricerche e trasferimenti tipici di un protocollo di file sharing.

DTNFile Ogni file simulato è un'istanza della classe *DTNFile*, che al suo interno contiene i campi necessari a descrivere il file stesso:

- Nome del file
- Hash che lo identifica univocamente
- Grandezza in bytes del file

Si è scelto di non inserire fra i campi anche le keywords che potrebbero descrivere il file e permettere una ricerca tramite parole chiave. Questo perché lo scopo delle simulazioni era quello di cercare e recuperare un determinato file già noto a priori, saltando quindi la prima parte di ricerca in cui l'utente può specificare più keywords secondo effettuarla.

DTNFileSystem I vari *DTNFile* in possesso di un nodo sono contenuti all'interno di un'entità chiamata *DTNFileSystem*. Come suggerisce il nome, il suo compito àš quello di simulare ad alto livello il file system del device rappresentato dal nodo. La realizzazione àš basilare, con i files condivisi contenuti in un'unica directory e delle funzioni che permettono le operazioni di:

- ottenere il numero di files contenuti
- verificare la presenza di un determinato DTNFile, utilizzando il suo hash come criterio di ricerca
- recuperare il riferimento di un determinato DTNFile
- recuperare una *Collection* contenente tutti i riferimenti ai files presenti nel file system
- inserire un nuovo *DTNFile* nel file system

Lo presenza del file system simulato, rappresentato da un'istanza della classe DTNFileSystem, all'interno di un determinato nodo àš dipendente dalla configurazione adottata per la simulazione, come descritto nella sezione 2.1. Per aumentare la flessibilità a seconda della simulazione desiderata, sono stati quindi aggiunti due parametri di configurazione:

Scenario.simulateFiles àš un parametro booleano che indica se simulare o meno la presenza di un file system nei nodi e la seguente distribuzione di files contenuti.

Group.fileCapability è un parametro booleano che indica se un determinato gruppo di nodi ha la possibilità o meno di utilizzare un file system per contenere dei files. Un gruppo di nodi rappresentante dei mezzi di trasporto pubblici, ad esempio, non avrà necessità di gestire la presenza di files al proprio interno.

DTNFileGenerator Per simulare la distribuzione di files fra i vari nodi, all'inizio della simulazione, è stato creato un generatore che, a partire da uno o più files di input, crea i vari *DTNFiles* e li distribuisce fra i nodi, a seconda di quanto configurato.
Ogni file di input può contenere più *DTNFileCreationRequest*, ognuna delle quali descrive il *DTNFile* da creare e come distribuirlo fra i nodi della simulazione. Una *DTNFileCreationRequest* contiene al proprio interno:

- tipo di distribuzione fra i nodi
- nome del *DTNFile* da creare
- dimensione in bytes del *DTNFile* da creare
- numero di copie da creare e distribuire
- nodi (singoli) a cui distribuire il *DTNFile*
- gruppi di nodi a cui distribuire il *DTNFile*

Gli ultimi tre parametri vengono utilizzati a seconda del tipo di distribuzione che dovrà essere applicata al file. I nodi interessati sono solamente quelli in cui è abilitata la simulazione del file system, cioè quelli appartenenti a gruppi in cui il parametro *fileCapability* ha valore *true*. I tipi di distribuzione possibili sono:

A (all): il file viene copiato per il numero di volte specificato e distribuito casualmente fra tutti i nodi

G (groups): il file viene copiato per il numero di volte specificato e distribuito casualmente solo fra i nodi appartenenti ai gruppi specificati

P (percent): il file viene copiato per la percentuale indicata relativa al totale dei nodi e distribuito casualmente fra tutti i nodi

H (hosts): il file viene copiato e distribuito solo fra i nodi selezionati. Se uno dei nodi indicati non ha attiva la simulazione del file system (*fileCapability* = *false*), viene lanciata un'eccezione gestita dal simulatore che provvede a terminare subito la simulazione informando l'utente dell'errore di configurazione.

Di seguito sono presentati alcuni esempi di configurazione, per la creazione e distribuzione di *DTNFiles*, che possono venire caricati dal *DTNFileGenerator*.

A mySong.mp3 3.5M 50

Indica di creare un *DTNFile*, di dimensione 3,5 MB e nome mySong.mp3, e di distribuirlo casualmente in 50 copie fra tutti i nodi con *fileCapability = true*.

P aPhoto.jpg 5M 25

Indica di creare un *DTNFile*, di dimensione 5.0 MB e nome aPhoto.jpg, e di distribuirlo casualmente fra il 25% dei nodi con *fileCapability = true*.

G aPhoto.jpg 2.4M 25 6 8 10

Indica di creare un *DTNFile*, di dimensione 2,4 MB e nome aPhoto.jpg, e di distribuirlo casualmente in 25 copie fra tutti i nodi appartenenti ai soli gruppi 6, 8 e 10

H ebook.pdf 650k 42 43 44

Indica di creare un *DTNFile*, di dimensione 650 kB e nome ebook.pdf, e di distribuirlo fra i nodi con indirizzo pari a 42, 43 e 44, una copia per nodo.

Come già accennato, è fondamentale che l'esecuzione di una simulazione sia riproducibile, sia per verificare i dati e i risultati ottenuti, sia per poterne variare un parametro, mantenendo però costante il comportamento del resto del sistema simulato. Questo ragionamento si applica anche alla casualità presente all'interno del *DTNFileGenerator*, ossia alla distribuzione delle varie copie dei *DTNFile* fra i nodi.
Per garantire che, effettuando più simulazioni utilizzando la medesima configurazione, i files vengano distribuiti agli stessi nodi, è stata adottata la stessa tecnica utilizzata dai modelli di movimento che estendono la classe *MovementModel*. In quel caso i moduli relativi al movimento vengono inizializzati utilizzando un parametro di configurazione relativo al seed per il *random number generator*. Nel caso di valori di configurazione e seed iniziale mantenuti immutati, i nodi si muovono nello stesso modo anche ripetendo più volte la simulazione: vengono infatti generati gli stessi valori relativi a destinazioni, velocità e tempi di attesa per i nodi.
If the seed and all the movement model related settings are kept the same, all

nodes should move the same way in different simulations (same destinations, speed and wait time values are used).

Creates a new random number generator using a single long seed. The seed is the initial value of the internal state of the pseudorandom number generator which is maintained by method *next*. In details, if two instances of Random are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers.

FileRequest A *FileRequest* represents the user request for a particular file. In every instance of this class are included:

- **fromAddr**: the address of the node operated by the user looking for the file
- **filename**: the name of the file searched
- **creationTime**: the simulated time when the request s submitted by the user

All *FileRequests* are read by the *M2MShareFileRequestReader* at the beginning of every simulation, then during the simulation, when simulated time become equal to the *creationTime* of the *FileRequests*, that is inserted into the correct queue of the node corresponding to the address included in the request.

M2MShareFileRequestReader It is the entity responsible for reading the *FileRequests* at the beginning and to dispatch the requests during the simulation, when the correct time comes. It implements the interface *EventQueue*, and so it provides the method *nextEventsTime()*, which returns when the next request will be ready to be inserted in the corresponding node, and the method *nextEvent()*, which returns the next request that will become active.

5.1.2 GUI

The GUI of THE_ONE has been modified to reflect the addition of file simulation support to the simulator. The detail window related to a node, show using the button R̈outing infoïow contains informations about the file system of the node, including all the complete files owned and the percent of VirtualFiles already downloaded.

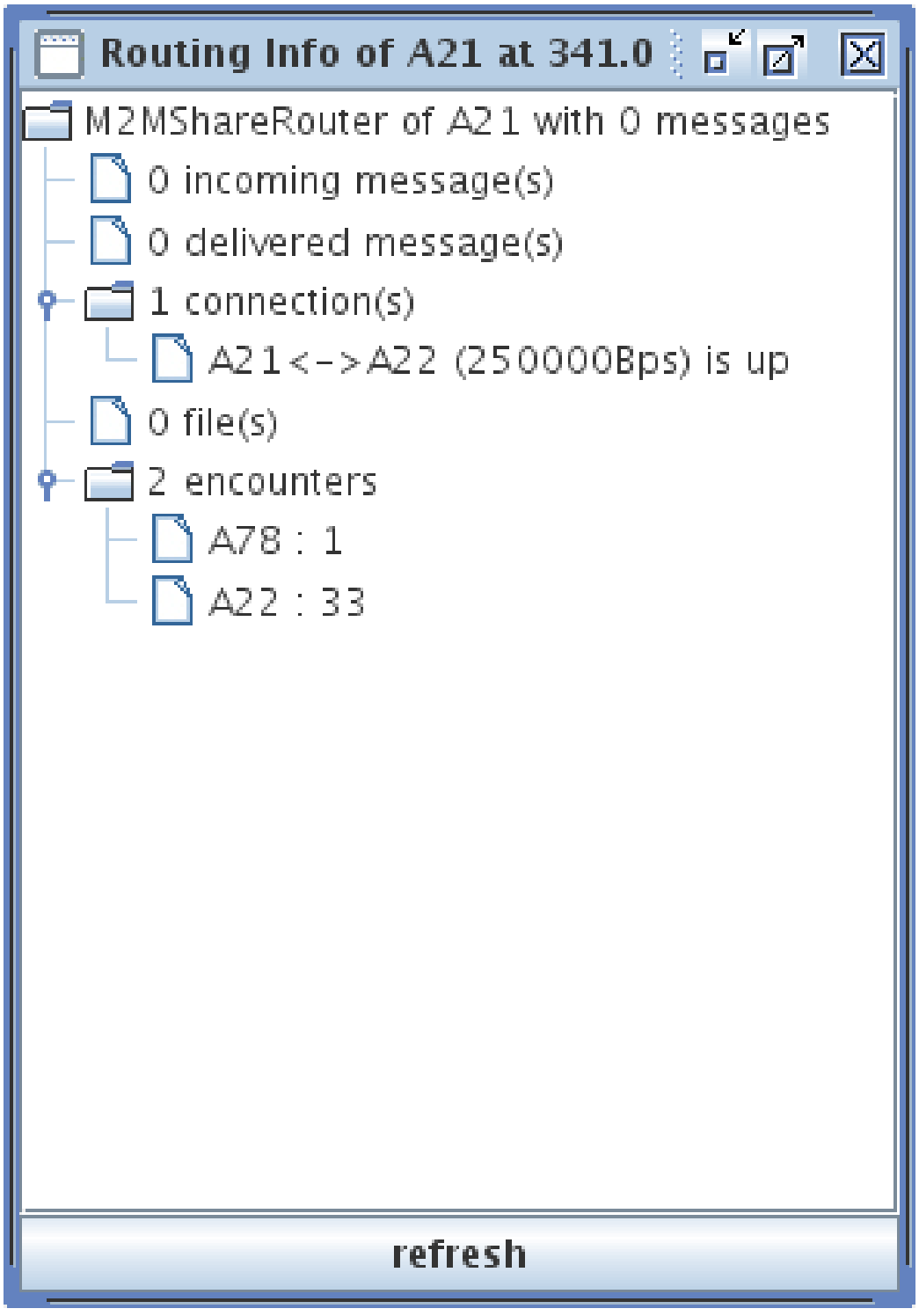

Figura 5.1: Routing Info - An example of window with a detailed view of a node state

5.1.3 Reports

A fundamental feature related to simulations is the capability of gathering data from different aspects of the simulated world, like positions of nodes, communications, data transfers and so on. To achieve this goal in THE_ONE are used some special modules named *Report Generators* which works with some related Listeners to catch the interested events in the simulated scenario and save them to some report files. The main objectives of our simulations was to gather informations about the efficiency of delegations and file division strategies, and to do so we collected data about several aspects for every simulation. We now describe the report modules created to the gathering of data during the simulations, specifying the type and mode of data collection used.

FileGatheringLog The first report module is a log report module, that store the selected events in a list, saving the to file one event per line. This module is responsible to listen for the following events:

- **VirtualFile creation:** a new *VirtualFile* has been created due to the execution of a *FileRequest* in a node. It emulate a user requesting a new file to be found and downloaded.
- **DTNPendingDownload creation:** a new *DTNPendingDownload* has been created in a servant peer due to the delegation of a task from a client node.
- **DTNPendingDownload completion:** a *DTNPendingDownload* has been completed in a servant peer (the servant has downloaded all the requested file interval subject of the delegated task). Consequently a *DTNDownloadFWD* has been created and enqueue in the servant.
- **DTNPendingDownload expiration:** a *DTNPendingDownload* has expired in a servant peer before it was completed (the servant has not downloaded all the requested file interval subject of the delegated task).
- **DTNDownloadFWD expiration:** a *DTNDownloadFWD* has expired in a servant peer before it was completed (the servant has not forwarded all the requested file interval subject of the delegated task to the requester node).
- **DTNDownloadFWD completion:** a *DTNDownloadFWD* has been completed in a servant peer (the servant has forwarded all the requested file interval subject of the delegated task to the requester node).

Every event is saved in the following format:

Sim_ time event_ description event_ details

The following are some examples:

15538.0000 PendingDownload A32 to E476

Is referred to the task delegation from the node A32 to the node E467, 15538 seconds after the start of the simulation.

38559.5000 PendingDownload completed in F630 (317200191 requested by A32)

Is referred to the completion of the *DTNPendingDownload* in the servant node F630 at time 38559.5. It was delegated by the node A32 and the id of the searched file was *317200191*

213738.0000 DownloadFWD expired in F523 (317200191 requested by A32)

Is referred to the expiration of the *DownloadFWD* in the servant node F523 at time 213738. It was delegated by the node A32 and the id of the searched file was *317200191*

DataTransferLog *DataTransferLog* module is another log module, which keep track of events regarding data transfer between nodes in the simulation. These can occur during the execution of several activities:

- **VirtualFile:** when the requester peer is in range with a node carrying the searched file, and the data transfer take place. Multiple data transfer events can be generated, if there are several peers with the searched file in range.
- **DTNPendingDownload:** when a servant peer downloads, from a node carrying the searched file, some requested intervals contained in the delegated task.
- **DTNDownloadFWD:** when a servant peer forwards the result of a delegated *DTNPendingDownload* to the client peer.

Data transfer events are saved in the following format:

Sim_ time from_ address to_ address data_ transferred (in bytes)

e.g.

98829.0000 G714 A17 2750001

Is referred to the data transfer from node G714 to node A17, 98829 seconds after the start of the simulation.

FileGatheringReport *DataTransferLog* module is the most important report module used during our simulations, because it summarizes all key aspects of a single simulation. In detail, these are the values tracked by this module:

- **Total data:** the total amount of data traffic exchanged between servant peers trying to satisfy a delegated task, file possessor peers and the data forwarding quantity toward the requestor node.
- **VirtualFile created:** number of *VirtualFile* task created during the simulation
- **VirtualFile delegated:** how many times a *VirtualFile* task has been delegated to a servant peer
- **PendingDownloads completed:** how many of the delegated tasks have been completed i.e. how many times the servant peer has been able to find and download all the data intervals requested in the *DTNPendingDownload*
- **PendingDownloads expired:** how many of the delegated tasks expired before the servant peer being able to find and download all the data intervals requested in the *DTNPendingDownload*
- **DownloadFWDs expired:** how many *DownloadFWDs* expired i.e. the servant peer downloaded all the data intervals requested in the *DTNPendingDownload* but it was not able to forward them to the requester peer before the task expiration
- **DownloadFWDs returned:** how many *DownloadFWDs* has been forwarded correctly to the requester peer

- **VirtualFile completed:** how many of the created *VirtualFile* has been completed
- **First VirtualFile satisfied:** the time (in seconds) passed from the beginning of the simulation before the first *VirtualFile* has been completed
- **Simulated time:** the simulated duration (in seconds) of the simulation
- **Simulation time:** the real duration of the simulation (in seconds)

Saving all these values for every simulation is useful to make some *a posteriori* analysis like averages, minimum and maximum values and to find best or worst case for a large number of simulations, as will be shown in section 6.

DelegationGraphvizReport The last report module implemented for our simulation is a module responsible for the creation of reports describing the delegation history of the simulation using a Graphwiz graph. These graphs are described using the DOT language and can be displayed using the *Graphwiz Graph Visualization Software*[1]. These graphs are useful to get a visual feedback about the delegations of tasks done and the status of that tasks, especially in case of multi-hop delegations. For every peer involved in delegations can be seen:

- if it receive the delegation (obviously)
- if it complete the delegated task
- if it forwarded back the result of the delegated task

In figure 5.2

5.2 M2MShare Implementation

In this section will be described the implementation of the modules composing M2MShare into THE_ONE. As said earlier in Section 2, every routing protocol is an extension of a common superclass named *MessageRouter* and, according to the THE_ONE philosophy, every new routing module can be inserted extending that class and it can be consequently used in configuration

[1] http://www.graphviz.org/

Figura 5.2: Graphviz graph example - An example of graph generated using the output of DelegationGraphvizReport module

time to set the routing behaviour of nodes.

M2MShare implementation consists in the main class extending *MessageRouter*, named *M2MShareRouter*, and several other classes contained into the package *routing.m2mShare*.

5.2.1 M2MShareRouter

As earlier said, this is the main class of M2MShare implementation. Its first responsibility is to load the related settings as set in configuration files (see Section 2.1) and initializes all the modules needed to the execution of the protocol. These are the entities which will be later described and are responsible of the different aspects of the protocol. Values used to initialize the settings are read from configuration files and are all included in the settings namespace *M2MShareRouter*. Every parameter have a default value, used in case it is not specified in any configuration file. Here are shown the available parameters and, shown in brackets, the relative default value:

- **M2MShareRouter.frequencyThreshold [2]** indicates the minimum number of encounter times needed for elect as servant a peer and consequently delegate to him a task

- **M2MShareRouter.scanFrequency [10]** indicates how many seconds the *PresenceCollector* have to wait between a scan and the next one
- **M2MShareRouter.delegationType [1]** indicates the type of delegation strategy used. It accept three values:
  - **0:** do not use delegation and file exchange is initiated only when a peer holding the requested data file is found in reach area
  - **1:** use the M2MShare technique where missing tasks are delegated only to peers which exceed the *frequencyThreshold* value
  - **2:** use the trivial technique where missing tasks are delegated to each encountered peer
- **M2MShareRouter.fileDivisionType [1]** indicates the type of file division strategy used. It accept three values:
  - **0:** for every file transfer is requested the entire file
  - **1:** use the M2MShare technique to choose the initial download point in the requested file
  - **2:** randomly choose the initial download point in the requested file
- **M2MShareRouter.useBroadcastModule [true]** used to enable/-disable the broadcast module. Could be useful to speed-up simulations, at the cost of a loss of precision
- **M2MShareRouter.delegationDepth [1]** indicates the maximum times a task can be delegated, starting from the initial requester
- **M2MShareRouter.stopOnFirstFileRequestSatisfied [false]** used to make the simulation stop when the first File Request has been satisfied. Can be useful in simulations in which we are not interested in what happens after the File Request satisfaction

The class *M2MShareRouter* also extends the superclass *MessageRouter* and doing so it override the main method of this class: *update()*. As said in Section **??**, that method is called one time for every simulated time interval, after the update of movement and connections of the node. In *M2MShareRouter* the only action executed in *update()* is to call the method *runUpdate()* in module *Scheduler*, described in section **??**.

5.2.2 PresenceCollector

5.2.3 QueuingCentral

5.2.4 Activity

VirtualFile

DTNPendingDownload

DTNDownloadFwd

5.2.5 Scheduler

Executor

Communicator

5.2.6 BroadcastModule

5.2.7 IntervalMap

Capitolo 6

La simulazione

Come ho implementato la simulazione
(forse meglio chiamare il capitolo Implementazione?)

Capitolo 7

Conclusioni

LE conclusioni di tutto

Bibliografia

[1] T.Kärkkäinen A.Keränen, J.Ott. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2009. ICST.

[2] Frans Ekman, Ari Keränen, Jouni Karvo, and Jörg Ott. Working day movement model. In *Proceeding of the 1st ACM SIGMOBILE workshop on Mobility models*, MobilityModels '08, pages 33–40, New York, NY, USA, 2008. ACM.

[3] A. Bujari. A delay tolerant solution for p2p file sharing in manets. Master's thesis, Università degli Studi di Padova, 2010.

[4] C. Lindemann A. Klemm and O. Waldhorst. Orion - a special-purpose peer-to-peer file sharing system for mobile ad hoc networks.

[5] S.Hong K.Lee S.Chong I.Rhee, M.Shin. Human mobility patterns and their impact on routing in human-driven mobile network.

[6] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[7] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007.

[8] Yang Zhang and Jing Zhao. Social network analysis on data diffusion in delay tolerant networks. In *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '09, pages 345–346, New York, NY, USA, 2009. ACM.

[9] F.Ekman. Mobility models for mobile ad hoc network simulations. Master's thesis, Helsinki University of Technology, 2008.

[10] Ari Keränen and Jörg Ott. Increasing Reality for DTN Protocol Simulations. Technical report, Helsinki University of Technology, July 2007.

[11] Brenton Walker, Masato Tsuru, Armando Caro, Ari Keränen, Jörg Ott, Teemu Kärkkäinen, Shinya Yamamura, and Akira Nagata. The state of dtn evaluation. In *Proceedings of the 5th ACM workshop on Challenged networks*, CHANTS '10, pages 29–30, New York, NY, USA, 2010. ACM.