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

Contents

Chapter 1

Introduzione

Una bellissima introduzione

Chapter 2

The ONE simulator

The simulation environment chosen for our simulations is The ONE (Opportunistic Network Environment) version 1.4.1[1], described in [1]. This simulation environment written in Java is completely configurable and is able to simulate completely the behaviour of nodes in the simulations, including movement, connections and to give a visual feedback about all these factors using its GUI. A more interesting feature, for our purposes, is the capability of emulate message routing using different routing protocols.

To emulate nodes movement, the ONE can take in input traces from real-world measurements, from external mobility generators or it can generate movement patterns using synthetic mobility model generators. Differences between several available mobility are explained in Section 3. Either mobility models that routing protocols are managed like independent modules, which are dynamically loaded depending on what set in configuration files. This allow a relatively easy implementation of new mobility models and routing protocols in the simulator.

The simulator finally allow to save data of interest from the completed simulations into report files. This reports are created using modules dynamically loaded in the same way to what happens with movement and routing modules.

2.1 Configuration

A single simulation is set, before running, using configuration files which describe the simulated environment, from the simulation length to number of nodes emulating people in the simulated environment. Configuration files

[1]The ONE simulator http://www.netlab.tkk.fi/tutkimus/dtn/theone/

are text-based files that contain parameters about the simulation, user interface, event generation, and reporting modules. All these parameters are loaded before the beginning of simulations and are used to adjust details about the behaviour of loaded modules.

Many of the simulation parameters are configurable sepa- rately for each node group but groups can also share a set of param- eters and only alter the parameters that are specific for the group. The configuration system also allows defining of an array of val- ues for each parameter hence enabling easy sensitivity analysis: in batch runs, a different value is chosen for each run so that large amounts of permutations are explored.

Inside configuration files, parameters are saved as key-value pairs and syntax for most of the variables is:

$$Namespace.key = value$$

Namespace defines the part of the simulation environment where the setting has effect on. Many, but not all, namespaces are equal to the class name where they are read. This convention is especially followed by movement models, report modules and routing modules, so also created new modules should follow it.

To make human-readability and configuration easier, numerical values can use suffixes kilo (k), mega (M) o giga (G), with "." as decimal separator. Boolean parameters accept "true" or "1", "false" or "0 ".

Comments can be inserted in setting files can contain comments too with "#" character. Rest of the line is skipped when the settings are read.

Every simulation can be set using several configuration files, to divide parameters into different categories, e.g. a file can contain scenario parameters, like maps, streets and districts, another file can contain parameters used to configure nodes, another file can contain reports configuration and so on. First configuration file read, if exists, is always "default_settings.txt". Other configuration files given as parameter can define more settings or override some (or even all) settings in the previous files. The idea is that you can define in the earlier files all the settings that are common for all the simulations and run different simulations changing parameters only in latests configuration files.

The basic agents in the simulator are called nodes. A node models a mobile endpoint capable of acting as a store-carry-forward router (e.g., a pedestrian, car or tram with the required hardware). Nodes in the simulation world are divided into groups, each one configured with different capabilities. Inside a group, every node has the same characteristics, i.e. are radio interface, persistent storage, movement, energy consumption and message routing protocol. It is possible to configure some of these capabilities as common for all groups, avoiding to set them individually for each group. Changing some configuration value between a group and another, allow to get heterogeneity between the behaviour of nodes in simulated scenario.

In configuration files is also possible to provide array of settings for every parameter. This allows to run large amounts of different simulations using only one single configuration file. Every simulation will be different from the previous and this is very useful to gather reports about different aspects in a common scenario.
Syntax for these configuration parameters is

Namespace.key = [run1_value; run2_value; run3_value; etc]

Some parameters, finally, accept a file path as value and this can be either an absolute or relative path. These parameters are used for maps, nodes paths or events to be loaded by event generators modules.

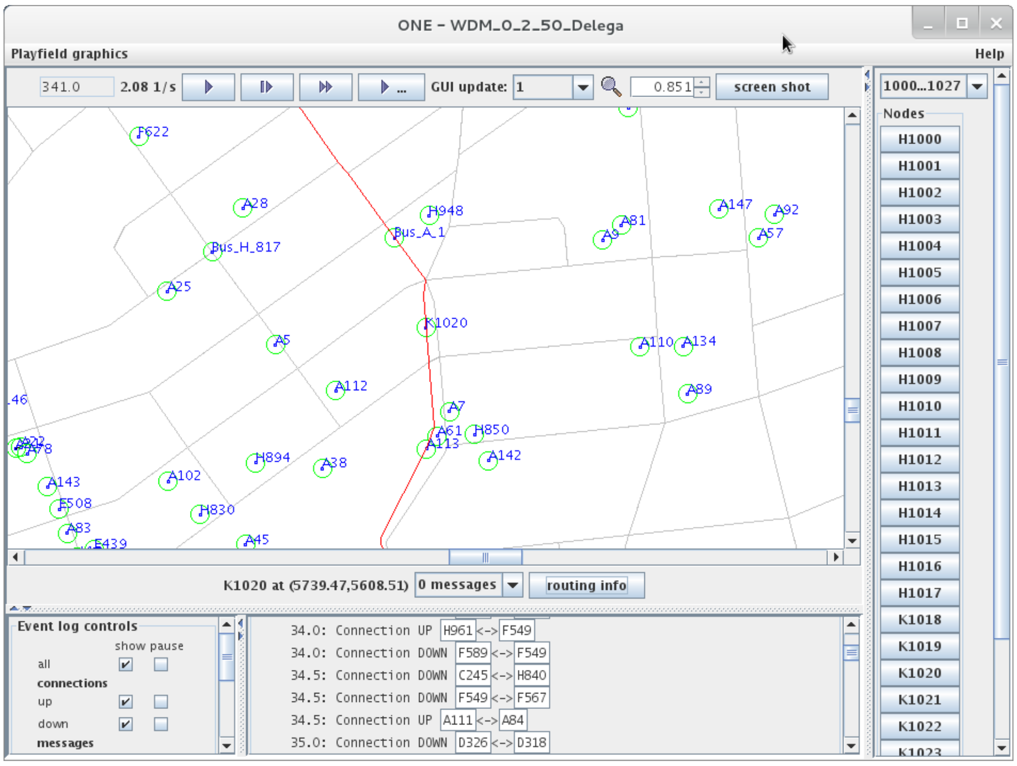
2.2 Simulation outputs

The way to follow the progress of simulations is the GUI present in the ONE. In main window, shown in Fig.2.1, is possible to observe movements of nodes acting in the simulation and view details about one of them by selecting it from the list of active nodes. For every node is possible to get informations about active connections, messages carried and other details. In the main window is also present a log panel updated with a textual description of events generated in the current simulation. These events can be filtered allowing to see only logs related to one kind of events e.g. new connections, creation of messages or massage exchanges.

Selecting a node is also possible to follow its movements inside the simulated world map, with the path to its next destination highlighted in the graphical visualization of the map. A pop-up window, shown in Fig. 5.1, can be invoked to get advanced informations about the state of that node.

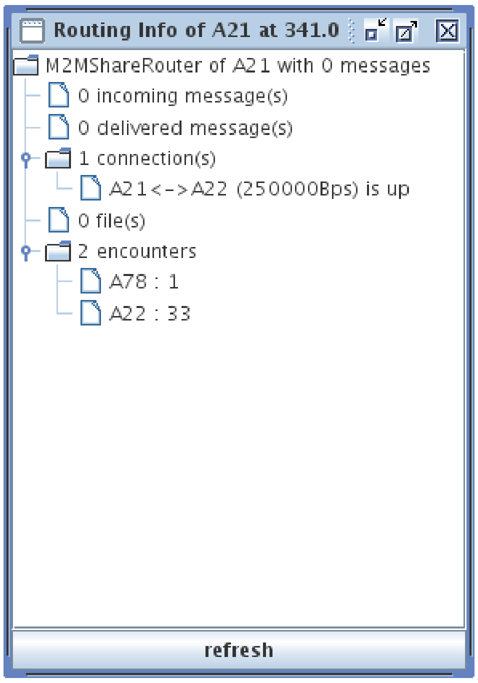
The map representation is configurable and the user can adjust the zoom level, simulation speed and it's possible to add an image as background, e.g. using a road map or a satellite photo of the interested area.

The other way to follow the progress of simulations is to read generated reports, at the end of every simulation. Each report is generated by its related module and these modules, like movement and routing modules, are



```
34.0: Connection UP H961<->F549
34.0: Connection DOWN F589<->F549
34.5: Connection DOWN C245<->H840
34.5: Connection DOWN F549<->F567
34.5: Connection UP A111<->A84
35.0: Connection DOWN D326<->D318
```

Figure 2.1: Main Window of the ONE


Routing Info of A21 at 341.0

```
M2MShareRouter of A21 with 0 messages
  0 incoming message(s)
  0 delivered message(s)
  1 connection(s)
    A21<->A22 (250000Bps) is up
  0 file(s)
  2 encounters
    A78 : 1
    A22 : 33
```

Figure 2.2: Example of detail window related to a node state

dynamically loaded at the beginning of the simulation according to what set in configuration files. To use report files is particularly useful when doing simulations in batch, without using the GUI and analysing resulting reports when all the set of simulations is complete. In batch mode is useful the possibility of define arrays of values for parameters in configuration files. These allow to set differences in configuration between the different runs of the batch. When the batch is executed, the ONE would run all the simulations, reading for each of them the correct configuration parameters, and saving the generated reports into different files for each simulation run.

Several report modules can be active during one single simulation. Output of these modules are report files, which typically are text-based files in which are saved statistical and log data gathered during the simulation. These data can be further analysed when the simulation has ended. In the ONE version 1.4.1, the simulator has some ready-to-use report modules which allow to create reports related to:

messages, e.g. number of created messages, delivery times, expired messages, etc..

contacts, which can show contact and inter-contact time between the nodes, or the number of contacts during the simulation

connections, which can show changes in connections status

Report modules are handled by the simulator like other modules and this allow to easily implement and add new report modules to the simulator, to gather needed data from simulations.

2.3 Simulation execution

It can be interesting to explain in detail how a single simulation is run by the simulator. Let's see what steps are performed for each run.

The first step done is configuration loading. The simulator read from configuration files values assigned to simulation parameters. Configuration files path are passed as arguments to the ONE launch command. When a configuration file is read, values contained are used to initialize or overwrite the value of some variables in simulation environment. If some parameters value is not set in configuration files, default values are used and an exception is thrown if a default value was not present for that parameter.

When the simulation is configured, the creation of the simulated scenario begins. The scenario contains all active entities in the simulation, i.e. nodes, report generators or event generators, and passive entities, i.e. maps which composes the simulated world. In this phase all nodes are created and initialized according to the configuration loaded. Every node contains a router implementing one routing protocol, uses one movement model and has some *listener* associated used to catch events and generate reports.

When all entities operating in the simulation are created an configured, the main phase of the simulation begins. This consist in an iterative update of the status of simulated world, obtained by calling a method *update()*, and increasing simulation clock value. This clock measure how much time is passed from the beginning of the simulation and incremental step value is configurable in configuration files, using the parameter *Scenario.updateInterval.* This accept a double value and it indicates how many seconds wait between an update of the world and the next one. This value have a big influence over results of the simulation. This because a simulation with a small *updateInterval* would be more accurate than one with a big value of update, having a more frequent update of the status of simulate world, but it also need more computational time to complete the simulation.

The first action executed in simulated world *update()* is to move nodes inside the simulation map. This is done according to every node's movement model and to *Scenario.updateInterval* value. E.g. a node simulating a car driver will move farther than a node simulating a pedestrian, considering the same *updateInterval* time value. As we will see in Section 3, related to movement models, these models can be very complex and can simulate several different movement behaviours, according configuration values used.

When it has been moved, for every node are updated connections and simulated router status. For every network interface is checked if the movement just occurred has changed its status, i.e. some connection has been closed because the two end points are now out of range or some new connection has been established. When the status of a connection changes, related *listeners* are informed, report module are informed too, if enabled, and GUI is updated showing changes in connection status, as shown in Figure 2.3.

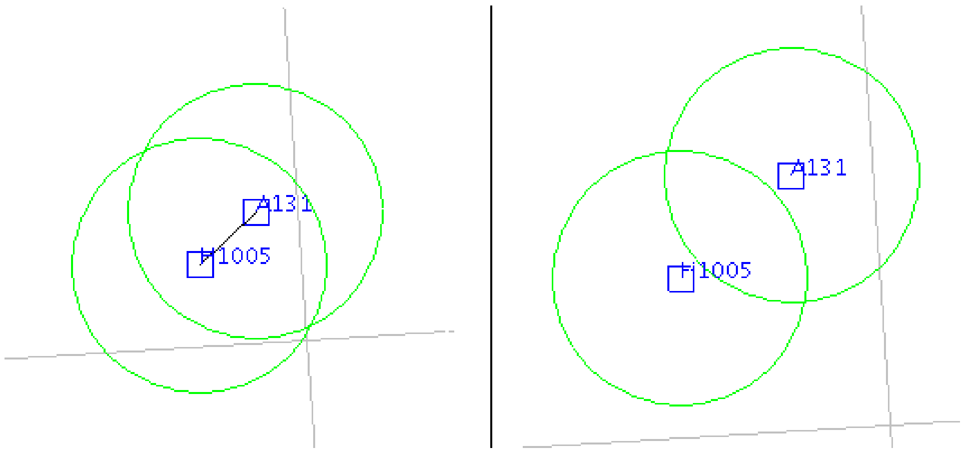


Figure 2.3: Connections - An example of graphical representation of connection between nodes. In the left side is shown an active connection, while in the right side nodes are no more in communication range, represented by the green circle around them.

Last action performed during simulated world *update()* is to update router status for every node. This part is strongly dependent on routing protocol simulated and, by overriding router module *update()* method, is the way to implement a new routing protocol in the ONE.

2.4 Randomness

One basic characteristic of a simulation done with a network simulator like the ONE is repeatability, i.e. running several simulations with the same configuration, results of these simulations will be the same, as will be the same the behaviour of acting nodes in the simulations. This is fundamental to verify correctness of data and results of completed simulations, or to repeat the simulation changing only few parameters over the total configuration, leaving constant the simulated world description.

To achieve a statistical correctness in analysis, we need to repeat a simulation several times, to get results independent from initial position and movement of nodes. Doing average analysis over simulations results, allow to study statistical trend of analysed variables, without focusing to best or worst cases that can happen during several simulations.

To achieve that result, each movement models modules uses a *random number generator*. This generator is initialized with a long seed, read from configuration files as value for the parameter *MovementModel.rngSeed*. 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 generator 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. This means that 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).

This is the strategy we used to get an high number of related simulations, from which we got results later analysed: we keep constant all simulation settings, specified by one configuration file and we change, for every run, *MovementModel.rngSeed* value. Doing so we got data to analyse, from reports, independent from initial position and movements of nodes.

2.5 The map

The map defines the space and routes in which the nodes can move inside the simulated world. The design of the map is an important part of the mobility model because it contains all the information of the locations of the houses, offices and meeting spots, as well as bus routes with bus stops. Since all the movement of the nodes is determined by activities with specific locations, the placement of these locations define how nodes are moving on a larger scale. Locations can be node group specific, i.e. some offices are used only by nodes in one group, nodes in different groups live in different houses locations, etc.. This makes it possible to create small districts within the map. In Fig. 2.4 is shown the default map included in the ONE simulator and used for our simulations and described in [**?**].

It is possible to see the four main district in which the map is divided.

In this implementation, every node group belongs to a district i.e. all nodes in one group will have their home location, office and meeting spot

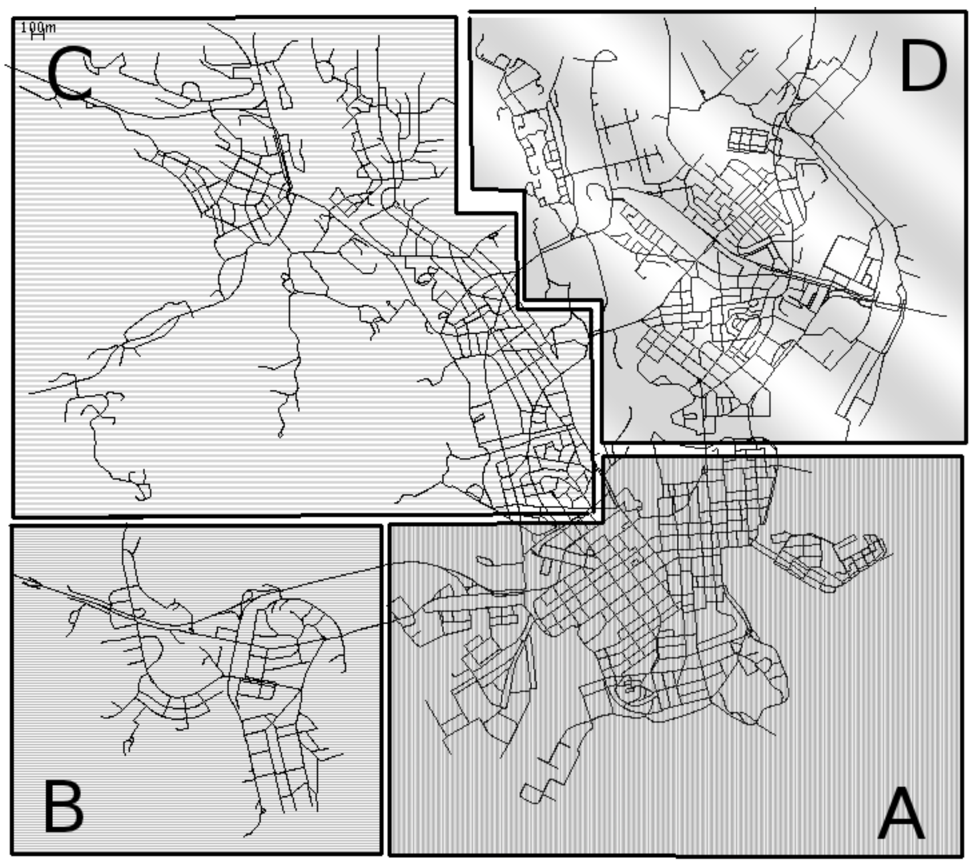


Figure 2.4: The map of the Helsinki centre area used in our simulations

inside the related district. This is useful to limit node movement to small areas. Doing this we increased locality and it is not possible that nodes belonging to different groups (and so different districts) can meet each others. To decrease locality it is possible to spread houses, offices and meeting spots on the map, having nodes meeting easily.

It is also possible to combine low and high locality: in Helsinki map used for our simulations there are four main districts, with different sizes, overlapped by other districts. This allows to have high locality but also some movement between districts, which corresponds to nodes coming to some district to work or meet friends, while others are leaving their district for similar reasons. Nodes moving between districts, not located next to each other, will have to pass through other districts. In Table 2.1 is shown distribution of nodes, offices and meeting spots in the districts of the map.

District	**Nodes**	**Offices**	**Meeting spots**
A	150	30	4
B	50	10	1
C	100	20	2
D	100	20	2
E (A + B)	100	20	2
F (A + C)	150	30	4
G (A + D)	150	30	4
H (Whole map)	200	40	5

Table 2.1: Table showing assignment of nodes, offices and meeting spots between districts in Helsinky map

Chapter 3

Movement models

in dtn no connessioni end to end mobilità dei nodi influenza protocolli di routing importanza di simulare il protocollo per valutare la rete globalmente modelli di mobilità usati per simulare movimento dei singoli users influenzano posizione e velocità desiderabile che simulino il comportamento dei nodi il più realisticamente possibile senza soffermarsi a simulare un singolo caso specifico

One important aspect to consider to achieve a good level of realism in DTN simulations is to simulate movement of nodes inside the simulated world. In Delay Tolerant Networks, movement of nodes is essential for the performance of the network, since end-to-end connectivity does not always exist and packets are delivered in a store and forward manner. For this reason is fundamental to simulate, with a good level of realism, movements behaviour of nodes inside the simulation.

Capturing movement accurately in the real usage scenarios is also needed for a reliable assessment of a new protocol. Movement can be captured in simulations using real movement traces or synthetic traces generated by a movement model.

3.1 Random Walk

The Random Walk movement model (RW) is one of the simplest mobility models available. In this model the simulation area is an open empty space where nodes move. Each node choose randomly an angle and a speed and for the next interval it moves using that speed and direction values. The choice is done for every simulation instant. The Random Walk model is a memoryless mobility model, because information about the previous status is not used for the future decision.

3.2 Random Waypoint

The Random Waypoint movement model (RWP) is a generalization of RW and is one of the most used in mobility simulations. In this model every node choose a random location inside the simulation area and moves toward it at a random speed uniformly chosen from $(V_{min}, V_{max}]$, where V_{min} is the minimum and V_{max} is the maximum speed of the simulation. When the node reaches its destination, it waits for a random amount of time and then repeats the previous steps. This model has the shortcoming, shown in [?], that the simulation do not reach a steady state and average speed of nodes constantly decreases if V_{min} value is set to zero, as in default implementation of the model.

3.3 Random Map-Based Movement

The simplest map-based mobility model is Random Map-Based Movement (MBM). Nodes adopting this model moves randomly but always in streets described in the map. This is done walking from one map node, i.e. an address on a street or a crossroad, to the other by always randomly selecting one of the directly connected map nodes. Result of this behaviour is a random movement in which there are more contacts between nodes than in Random Walk model. This is due to the restriction to node movement to follow streets described in the map.

3.4 Shortest Path Map-Based Movement

Shortest Path Map-Based Movement (SPMBM) is another mobility model that uses a map-described environment to restrict node movement. With this model, nodes chooses their destination randomly inside the map, then calculate the shortest path to reach it and finally walk along this path. Dijkstra's algorithm is used to calculate the path. Destinations can be chosen in a totally random way or from a set of Points of Interests (POI). This can be useful to emulate interesting places like restaurants, monuments or shops. This model adds one level of realism, compared to MBM, since nodes don't move in a completely random way, but they have destinations and follow the best path to reach it.

3.5 Routed Map-Based Movement

A movement model in which nodes movements are completely determined is Routed Map-Based Movement (RMBM). In this model nodes moves following predetermined routes, for the duration of the simulation. Nodes adopting this model can easily emulate the scheduled and repetitive movement of public transport, e.g. buses, trams or trains.

3.6 Working Day Movement Model

Movement models previously described are simple to understand and implement, but they don't completely capture such characteristics like heterogeneous behaviour, relationships between users or repetitive behaviour. Working Day Movement Model (WDM) [2] does this using a simple idea: people follow certain routines during a typical working day, i.e. go to work, go out with friends, etc.

All these actions make a person to meet usually the same group of people. This reflect heterogeneity in user movement, i.e. one node will hardly meet all other nodes in the simulation. This also raise locality in nodes movement, reducing encounters between nodes.

During a typical working day emulated by this model, a node is involved in three main activities:

- sleep at home
- work in the office
- go out in the evening with friends

These activities can obviously vary from person to person, depending on the lifestile of the subject. These are the most generic and can be associated to the typical working day of a large amount of people.

A person usually does these activities in the same locations, i.e. it sleep in its home and works in the same office. It can be assumed that when it goes out with friends, they will go to a set of favourite places. These activities are repetitive, day by day, and are performed in places in common with other people. Simulating this, allow the spontaneous creation of users communities, e.g. people living in the same home location are a family, users working in the same office are colleagues and friends will meet to go together to their favourite places in the evening. A movement model able to emulate all these behaviours is fundamental to evaluate a protocol like M2MShare, where the frequency of encounters between nodes influences the routing strategy.

Creation of such communities of users, and in general relationships between nodes, are not shown using simpler movement models like RMBM or SPMBM i.e. in a longer period of time, each node is equally likely to meet any other node an equal number of times. In the Random Walk movement model, nearby nodes are more likely to get in contact and there are no clear relationships between nodes in most of the simple models.

To emulate the three main activities, WDM uses simpler mobility models as submodels. A node moving using WDM, will change several submodels during a single simulated day. Submodels are also used to simulate different modes used by nodes to move through the map. A person can so move from a location to another walking, driving his car (if it has one) or using public transport.

3.6.1 Example of working day

During one typical working day, the starting location of a node is its own house. Every node has one waking up time, indicating what time that node leaves home in the morning. The value is drawn from a normal distribution with mean 0 and standard deviation configurable. The wake up time value is generated at the beginning of the simulation and remains the same for all the simulation duration, i.e. every node leaves home at same time every day. Different values between nodes emulate differences in lifestyles, e.g. one person can be faster to get dressed in the morning and another can be slower.

When a node leaves home, it travels with destination its office. This can happen walking, driving its own car (if available) or using public transport. The choice of transportation is done evaluating availability and speed to reach the destination. Accordingly to the chosen transportation is used the relative mobility submodel.

Once the node reaches its office, it stays there for all the working time, which is a configurable value. After the working hours, the node decide, by drawing, whether it returns home or goes out for the evening activity. Different submodels are used again for transitions between the locations.

3.6.2 Home Activity Submodel

Every node has one location in the map set as Home Location. Home location is randomly chosen, for every node, at the beginning of the simulation between a set of geographical points in the related district of the map. The chosen location remains the same for all the simulation duration, i.e. every node wake up in the morning and returns in the evening to the same location.

When the node returns home it moves near this location for a while and then it stays still until next wake up time. This behaviour is not an error, but it is done to emulate the activity to left the mobile phone in the same place inside the house while the user does typical domestic activities which do not involve the use of mobile phone, e.g cooking, watching TV or sleeping.

3.6.3 Office Activity Submodel

The submodel related to the working activities is a model which emulate the behaviour of an employee inside its office. During the working time the employee stays at his desk for most of the time but it can leave it for attend a meeting, talk with a colleague of have a break. During all these moments it can meet nodes related to other colleagues.

The office is described as a single room with rectangular shape, where the only entrance point, the door, is placed in the upper left hand corner. Every employee has its own desk, located in the same place inside the office for all the simulation duration. In this model is not simulated the presence of walls or other furniture inside the office. To compensate for this, the office is described as a room bigger than the normal size, to emulate time needed to avoid obstacles and reach a destination when moving inside the office.

When entered in the office, the employee moves directly to its desk and then stays there for an amount of time drawn from a Pareto distribution. When this period is passed, the node chooses a random destination inside the office, walks to reach it and then waits for an amount of time drawn from the same Pareto distribution, before to return to its desk. The repetition of this movements and pauses from the desk to some random locations inside the office continues until working time expires.

The working time length is a configurable parameter, as are configurable distribution parameters. This allow to tune movements inside the office as there are a variety of different jobs and buildings where people move according to different patterns e.g. a teacher which moves every hour to change class inside a school or a cashier which do not leave its place for all the working time.

3.6.4 Evening Activity Submodel

The Evening Activity Submodel emulates activities performed after work, in the late afternoon or in the evening, like shopping, go eat in a restaurant or in a pizzeria. These activities are performed in group and with a configurable probability which determines if a person returns directly home, after work, or stays out with friends.

When working time is expired, the node moves to one of its favourite places using one transport submodel. When arrived, it waits for other people until there are enough people to create a group and start the activity. The minimum number of people needed to create a group is configurable and when all groups for a place are full and a new person arrives, a new group is created.

When a group is full, all people in the group walk together to reach a near, randomly chosen, location and they wait for a random amount of time, drawn from a range of values. This waiting time emulates the duration of the dinner, shopping or a movie at cinema and when it is over all nodes belonging to the group leave each other and return to their own houses.

3.6.5 Transport Activity Submodel

Transport Activity Submodel used by nodes to move between home, office and evening activities.

At the beginning of the simulation, to every node is assigned a car with configurable probability. Nodes with cars moves always using it, while people which do not have one walks or uses public transport. A model which uses different types of transport models adds additional heterogeneity and this influences also routing protocols, since quicker nodes, like ones using cars for instance, can quickly transfer packets for longer distances.

According to the chosen transportation, Transport Activity Submodel uses three different submodels:

Walking Submodel : nodes which do not have a car walks on the streets toward the destination at constant speed. They uses Dijkstra algorithm to find the shortest path from the current position to the destination.

Car Submodel : nodes with a car move faster than walking nodes, driving between a main activity and the next one, but walks during a single activities, i.e. they do not use the car inside the office, of course. In this model are not considered traffic congestions, traffic lights and every car carries one single person.

Bus Submodel : there can be several routes of public transport (buses, trams, trains) in the simulated map. Each route is run by several public transport according to a schedule. Each public transport can carry more than one person at a time.

Every person which do not have a car knows one bus route and can use every bus running on that route. The choice between walking or using public transport is done considering a difference between Euclidean distances.

If the distance between node's location and the nearest bus stop summed with the Euclidean distance from the destination to the nearest bus stop is shorter than the Euclidean distance between the node's location and the destination, than the node uses the public transport system, otherwise it walks to the destination. If public transport is chosen the node changes between two transport submodels: it walks to the nearest bus stop using Walking Submodel and, switching to Bus Submodel, it waits for the bus. When the bus arrives, the node enters it and travels until to the bus stop closer to the destination. Finally it switches back to the Walking Submodel to reach the destination.

3.7 Real user traces VS Synthetic models

The most realistic user movement or contact patterns are the ones that happen in the real world. Therefore, several studies have focused on tracking user movement to be able to use the traces later directly for simulations or to learn more about which characteristics are common in real user behaviour. Real movement traces have usually been obtained by analysing WLAN access point data or having users carry around devices equipped with GPS modules. Contact traces have mostly been collected from real world experiments where users have been carrying around Bluetooth devices tracking other devices within range.

There are some problems about directly using the collected traces in simulations. Real world traces are usually from very specific environments like university campuses areas. These traces are not so generic to be used to simulate movement in a different environment, like a centre of a city, because behaviour and movement patterns of interested people are different.

Real trace are generally been obtained by analysing WLAN access point data or having users carry around devices equipped with GPS modules. In the first case only certain areas have WLAN coverage and so movements outside access points communication range are not recorded. Using GPS modules limits mobility recording only to outdoor movements.

When the locations of users are derived from access point data, the result is roughly building level granularity [16]. This is because users are not always connected to the closest access point and the movement between access points is difficult to capture. Most of the devices connecting to WLAN access points in these experiments are laptops and PDAs which are not always carried by the users and are not necessarily always turned on. Whether this characteristic is wanted in the simulations depends on what is modeled and simulated. Paper [?] argues that the on/off times are an important characteristic of wireless users that needs to be taken into account and modeled for simulations.

It is also practical to have a model with configurable parameters to work with. Sometimes a protocol or an application developer wants to test how the protocol or application performs when certain parameters change, to better be able to find out weaknesses and strengths. Sometimes researchers also need a very simple model to work with, to be able to use it in mathematical proofs for various theorems.

Synthetic models are often preferred since they are easier to work with than real user traces. Moreover, real user traces are rarely available for the environment to be modelled. Additionally, researchers want to do sensitivity analysis to find out how protocols and applications perform under different conditions. This is not possible with real user traces unless a parameterized model has been successfully extracted. There are two types of synthetic movement models that have been proposed for these analyses — generic high level models that aim to produce movement accurate enough with statistical measures, and models that describe incidental scenarios, hoping for a more accurate depiction of single devices. While efficient to use in simulations, the high level models, such as Random Waypoint [18], often imply that the scenarios for which the protocols are simulated have huge numbers of nodes, so that the relevant protocol features are given statistically realistic distributions of events. For scenarios with few nodes, the differences between different usage scenarios become more significant. Thus, movement models that depict more precisely some specific types of movement are needed. Traditionally, the approach to create a movement model has been to identify a certain characteristic of mobility and to create a mathematical model describing the movement at a high level. Such characteristics can be speed distribution, social relationships between nodes or favorite locations nodes will visit. These types of movement have very few details and the movement is homogeneous in the sense that every node is moving according to the same rule.

Before we go any further, we introduce two new terms: locality and heterogeneity. By locality we mean the tendency of nodes spending most of their time within a small area. Thus, a movement model where each node's movement is restricted to a small area has high locality. By heterogeneity, we mean different movement patterns and properties between nodes. Measuring heterogeneity in a specific context is not straightforward, but in our experiments we will later on talk about differences in contact patterns in terms of how often a node encounters new nodes compared to the fraction of earlier encountered nodes.

Hsu and Helmy [26] show by studying real user traces that nodes are very often turned on/off and only visit a small portion of the WLAN access points in campus areas. Moreover, they find that node mobility while using network is very low and one node only meets a small portion of all other nodes in the area. These types of characteristics are usually not captured in movement models. Furthermore, they reveal repetitive patterns with a period of one day and heterogeneity among nodes. Although heterogeneity and repetitiveness has been modeled, most simple movement models do not. According to Hsu and Helmy, the biggest issue with most synthetic models is that they are not capturing such characteristics as heterogeneous behavior, switching devices on/off or relationships between users. We noticed that even though most of the known features of movement have been modeled, no model exist where all or many of the features have been combined. Our approach is to combine these different elements to create a new movement model.

Chapter 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 3.6 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

4.6 File Division Strategy

Chapter 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 **??**. 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).

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

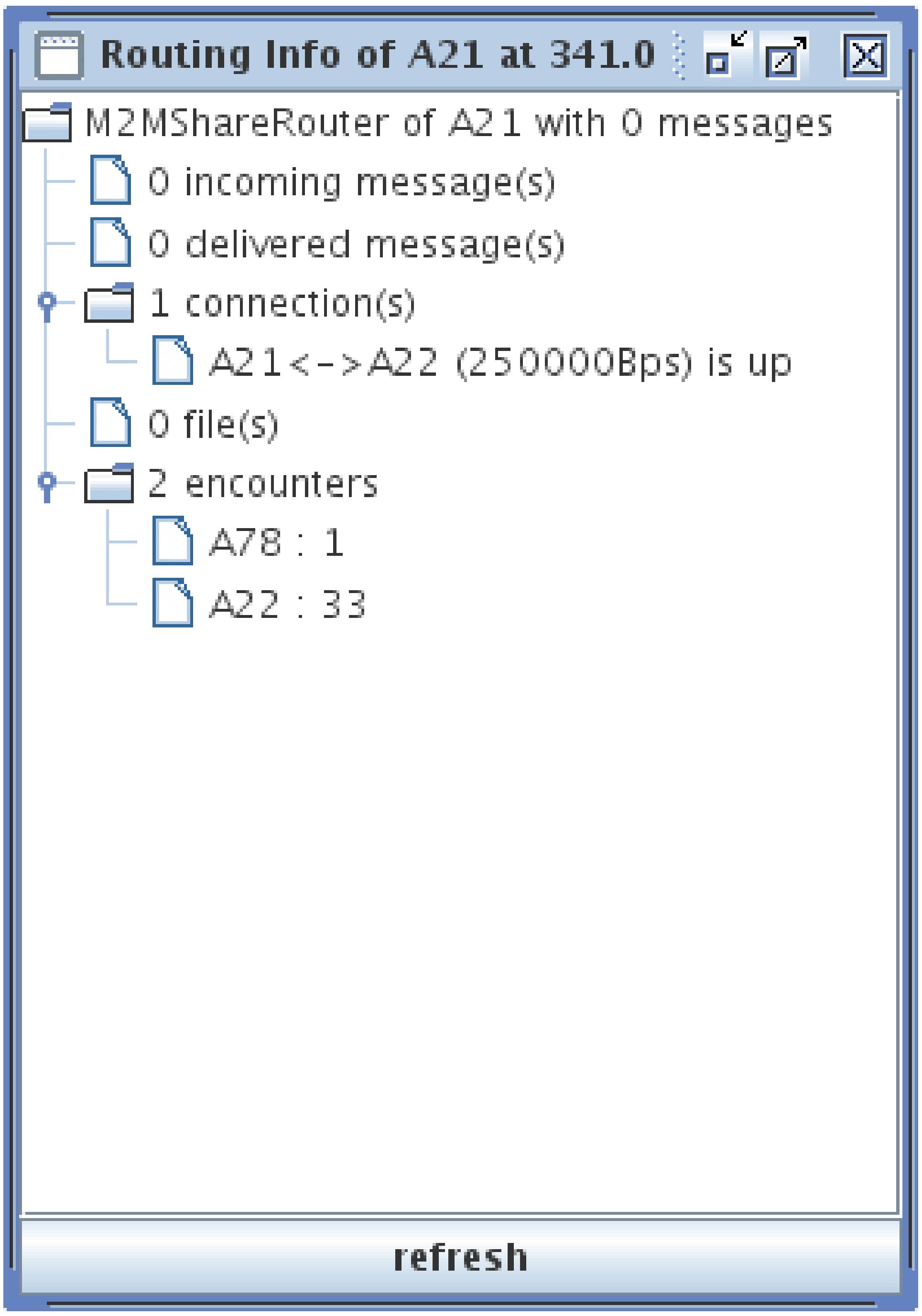
The GUI of THE_ONE has been modified to reflect the addition of file simulation support to the simulator. The detail window related to a node, show using the button R̈outing infon̈ow contains informations about the file system of the node, including all the complete files owned and the percent of VirtualFiles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

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.


Routing Info of A21 at 341.0

```
M2MShareRouter of A21 with 0 messages
  0 incoming message(s)
  0 delivered message(s)
  1 connection(s)
    A21<->A22 (250000Bps) is up
  0 file(s)
  2 encounters
    A78 : 1
    A22 : 33
```

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

- **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 **??**.

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

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

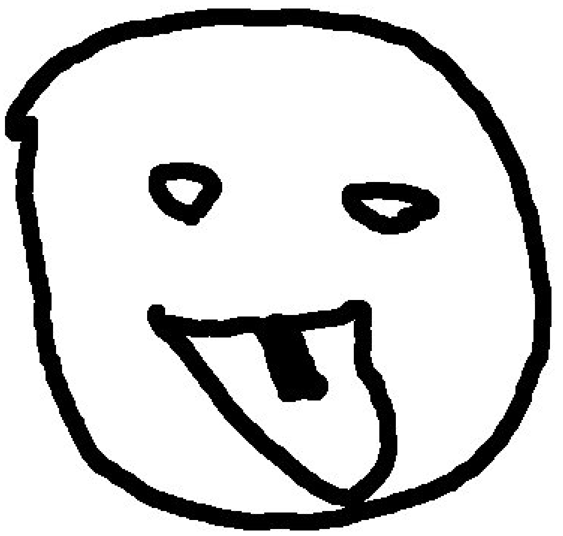


Figure 5.2: Graphviz graph example - An example of graph generated using the output of DelegationGraphvizReport module

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 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 2.3, 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 5.2.5.

5.2.2 PresenceCollector

The module *PresenceCollector* is responsible of gathering information about in-reach area devices and to track encounters between other nodes to realize the election strategy selected for the current node in the simulation.

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

Chapter 6

Simulations and results

In previous chapters we focused on the concepts, related technologies and implementation realized of the protocol, M2MShare, subject of the study of this thesis. In this section we describe the simulations executed and analysis with relative results. For every type of analysis we describe the related simulations, the configuration of the simulated world and the other protocols compared to our M2MShare. Finally, for every analysis, we show the results in a graphical way with a textual interpretation.

We repeat each of the simulation scenarios several times in order to achieve more accurate results, independent of the initial positioning of the requester peer in search for that particular data file and independent of the initial positioning of the searched file copies. Each scenario is run using a different random seeds to initialize the movement model and for every seed the simulation is repeated using the three compared protocols.

Before proceeding further, we introduce the reader to the adopted terminology and some "must know" configuration parameters:

- *Population*: refers to the number of nodes in the simulation which emulate people operating M2MShare. This number don't include nodes which emulate public transport, like buses or trams. People are distributed in districts of the map, described in Section 2.5.
- *File size*: refers to the size of the data file the user is looking for in the simulation
- *File popularity*: refers to how many copies of the data file the user is looking for in the simulation are present at the beginning of the simulation. Nodes initially carrying the file are randomly chosen, using the DTNFileGenerator described in Section 5.1.1 and can be uniformly chosen between all the active nodes or in a subset of them.
- *Delegation Type*: refers to the type of delegation used in the group of simulations. It can be of three types:

  - **No_delegation:** do not use delegation and file exchange is initiated only when a peer holding the requested data file is found in reach area
  - **M2MShare:** use the M2MShare technique where missing tasks are delegated only to peers which exceed the *frequencyThreshold* value
  - **Delegation_to_all:** use the trivial technique where missing tasks are delegated to each encountered peer

- *Delegation Depth*: how many times a delegated task can be re-delegate to other peers.

- *File Division Strategy*: refers to the type of file division strategy used during file transfer between nodes. It can be of three types:

  - **M2MShare:** use the M2MShare technique to choose the initial download point in the requested file
  - **iM:** for every file transfer is requested the entire file
  - **rM:** randomly choose the initial download point in the requested file

- *Nr. of simulations*: how many times the simulation has been repeated with different movement and file generation seeds.

- *Simulated time*: the simulated length of simulations

To increase the trustworthiness of the experimental outcome we have included realistic day-by-day node movements through the Working Day Movement (WDM) model, described in Section 3.6. As anticipated, this model is able to represent both the unpredictability of certain movements of users and the routine of other movements such as, for instance, the daily trip from home to work. This provides a good approximation of inter-contact times and contact durations, providing the flexibility for configuring real life test scenarios.

We use the same configuration parameters described in [2], to initialize the movement model: the map is the one described in Section 2.5, in which half of all the nodes were set to travel by car. When a node is walking, the speed is set between 0.8 and 1.4 m/s and for buses between 7 and 10 m/s, with a 10 - 30 s waiting at each stop. The speed of the cars is set to 20 m/s to make it a faster way to move between locations. Every node has a probability of 0.5 to do some evening activity after work, with groups size variable between 1 and 3 nodes. The working day length is set to 28800 s (8 hours) and the pause times inside the office is drawn from a Pareto distribution with coefficient 0.5 and minimum value 10 s. The office size is set to a $100m x 100m$ square, to compensate for the lack of floors, walls and other furniture.

6.1 Delegation efficiency

Population	1000
File size	3.0 MB
File popularity	50 copies uniformly chosen
Delegation type	No_delegation, M2MShare, Delegation_to_all
Delegation depth	1
File Division Strategy	M2MShare
Nr. of simulations	40 x 3
Simulated time	One week

In this analysis we evaluate the efficiency in using delegation versus not using it. We compare the efficiency of our system (M2MShare) employing delegations against two other systems using different strategies:

- *No_delegation:* system which does not employ delegations and file exchange is initiated only when a peer holding the requested data file is found in reach area of file requester.

- *Delegation_to_all:* system employing delegations but instead employs the trivial technique where missing tasks are delegated to each encountered peer.

The metric we study is the found time (Ft) for a generic data file which is the time interval between the first delegation made and the time an output return for that specific file is received. If no delegations are made and the first file request is satisfied by a direct file possessor the Ft is equal to zero. For each of the above scenarios we also measure the:

- number of delegations used: representing the number of tasks the requester peer has delegated for that particular data download;

- percentage of completed task: representing the number of delegated tasks completed (output returned) over all delegated tasks;

- total data transferred: referring to the quantity of data traffic exchanged between servant peers trying to satisfy a delegated task, file possessor peers and the data forwarding quantity toward the requester node.

In Fig. 6.1 it is possible to see the advantage, in terms of found time, in using the delegation technique instead of not using it. The two systems employing delegations find the required file in less time in each simulation run at the expense of higher overhead in terms of bandwidth due to delegations.

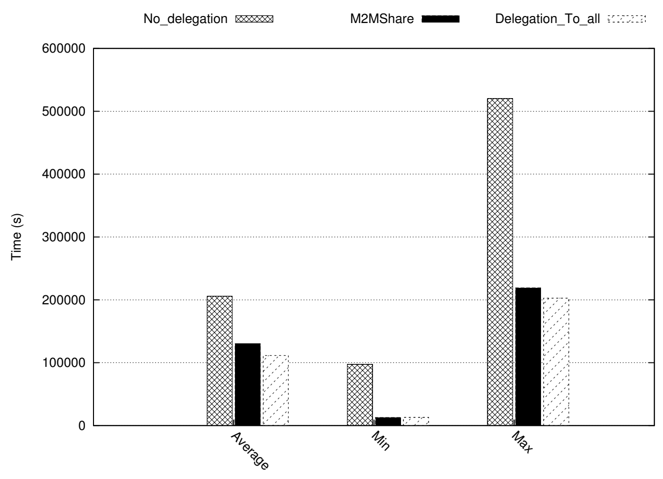


Figure 6.1: Average, min. max found time employed by each strategy in finding the required data file.

The system employing delegations to all encountered peers gets a better result on average, but at a cost of a higher number of delegated tasks (Fig. 6.2). A higher number of delegated tasks imply more bandwidth used for searching the data file and potentially retrieving (if found) and forwarding it toward the requester.

Fig. 6.2 makes a comparison between the two systems employing delegations by showing the number of overall employed task delegations till file download or simulation time expires. It is easy to see that M2MShare uses fewer delegations while achieving a higher percentage of completed delegated tasks (Fig. 6.4). This outcome is due to a conservative delegation strategy employed by M2MShare in delegating unsatisfied, unaccomplished tasks only to frequently encountered peers (servants). Since we do not have any means to evaluate the ability of one servant to satisfy a file request what we do is delegate to encountered peers whom is expected to be encountered again in the future. The Delegation_to_all strategy contributes to higher overhead also due to completed tasks, ready to be returned toward the requester that unfortunately expire and are discarded before having the chance of encountering the data file requester.
It is also possible to see in Fig. 6.3 that in M2MShare it takes less time, on average, for a servant to return the output of a delegated task which has been completed, i.e. not expired, to the requester node.

In Fig. 6.5 is shown the comparison between M2MShare and Delegation_to_all strategies in terms of transferred data quantities till file download or simulation time ends. It is straightforward to notice the higher overhead in terms of data transmissions introduced by delegating to all en-

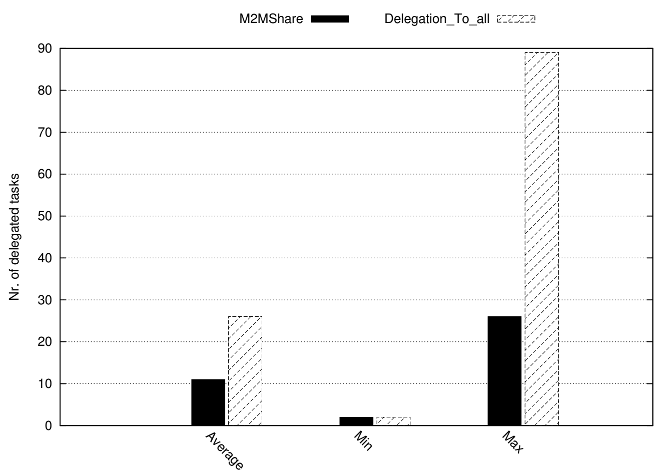


Figure 6.2: Average, min, max number of delegations employed by each delegation strategy.

countered peers whether M2MShare reduces the exchanged data quantity while still achieving the goal of acquiring the requested file. From the above results it is obvious that the delegation strategy serves its purpose by extending a peer reach area to other mobile disconnected networks where data content might be available therefore reducing the found time of a desired content. Although this strategy introduces an overhead in terms of bandwidth usage, computation and power consumption we control these side effects by delegating only to frequently encountered peers whom are expected to be encountered again in the future.

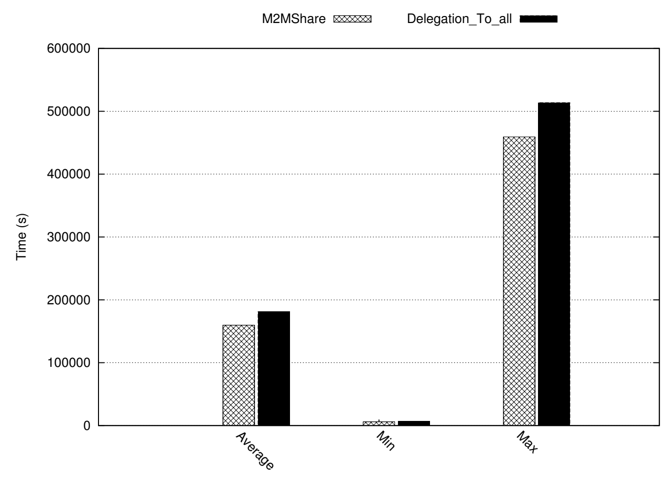


Figure 6.3: Average, min, max output return time for systems using task delegations.

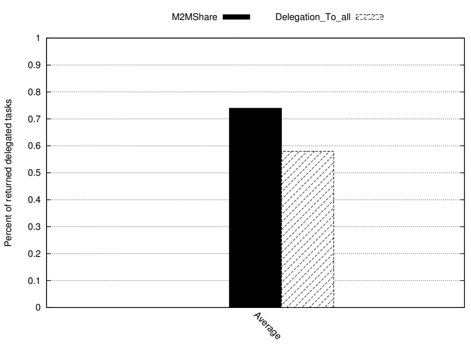


Figure 6.4: Percentage of completed previously delegated tasks against the number of overall delegations employed.

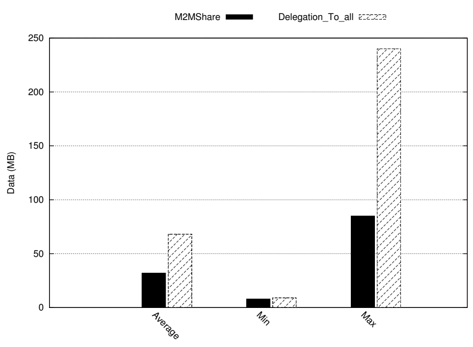


Figure 6.5: Average, min, max transferred data amount employed in each delegations technique.

6.2 Delegation efficiency with variable file popularity

In previous analysis we show the advantage, in terms of found time of the searched file, using M2MShare delegation strategy against avoid using delegation or delegate unaccomplished tasks to every meet node. The confrontation was made using a constant number of copies of the searched multimedia file uniformly distributed between all nodes in the simulation, i.e. 50 copies. The current analysis want to show the difference in performance using the three delegation strategies changing the initial file popularity of the searched file. To this aim, in these simulations we change the File popularity (Fp) of the multimedia file keeping constant the number of total nodes (N).

Population	1000
File size	3.0 MB
File popularity	50, 100, 150, 200, 250, 300, 350, 400 copies
Delegation type	No_delegation, M2MShare, Delegation_to_all
Delegation depth	1
File Division Strategy	M2MShare
Nr. of simulations	40 x 8 x 3
Simulated time	48 hours

We change the File Popularity (Fp) value, from 5% (50 copies) to 40% (400 copies). When the Fp is low (with Fp £ 5%) the system which don't use delegation is not able to find any piece of the file during the simulation time. We have indicated this in the chart by assigning to Ftavg a value of 48 h. With higher values of Fp, the first protocol is able to find the file thanks to the higher popularity of the requested file, but it takes more time than M2MShare and Delegation_to_all. Finally, with the highest values of Fp the performances of the three compared systems became very similar.

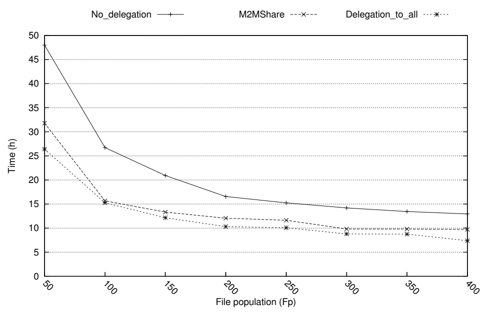


Figure 6.6: Percentage of completed previously delegated tasks against the number of overall delegations employed.

6.3 Delegation efficiency with variable nodes population

Population	100, 200, 400, 600, 800, 1000
File size	3.0 MB
File popularity	5%, 10%, 50%
Delegation type	No_delegation, M2MShare, Delegation_to_all
Delegation depth	1
File Division Strategy	M2MShare
Nr. of simulations	40 x 6 x 3 x 3
Simulated time	48 hours

In previous analysis we considered as constant the number of nodes emulating people using M2MShare on their devices. In the current analysis we changed the total population of the simulations, observing how this affects the performance of compared systems.

In the first scenario (Fig. 6.7) we consider Fp = 5%. The protocol not employing delegations (black line in the chart) is not able to find any piece of the file during the simulation time when the considered nodes in the scenario are equal or less than 400. We have indicated this in the chart by assigning to Ftavg a value of 48 h. This is due to the trivial strategy employed by the protocol and to the sparse environment. Even when able to find some node possessing the file (with $N \geq 600$), the time needed results bigger than using a strategy implementing delegations. Clearly, when increasing the file population (Fp), even the number of nodes in the population that posses the data file increases; as a result, the time to retrieve the file decreases for all solutions. A similar result is achieved also when considering a wider popularity for the required multimedia file (Fp = 10%, in Fig. 6.8). However, in this case, the higher popularity of the requested file helps both solutions in finding the file possessor with a smaller Ftavg than in the previous scenario. Finally, in Fig. 6.9, the performances of the compared solutions are very similar. This is due to the high file popularity among nodes (Fp = 50%): the chances of eventually finding a file possessor in a short time are clearly much higher.

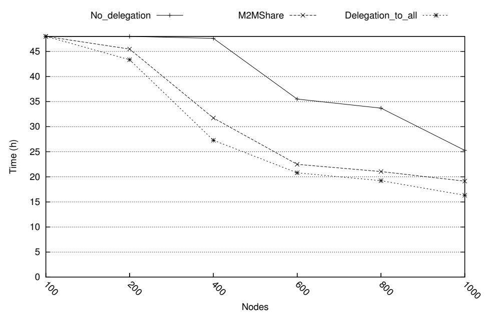


Figure 6.7: Average found time with $Fp = 5\%$

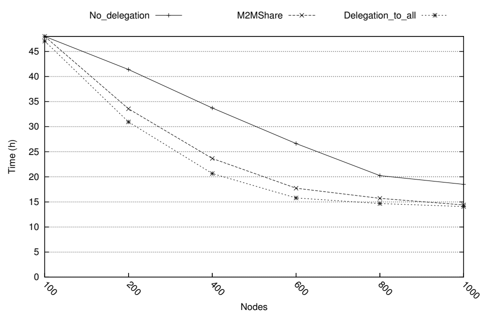


Figure 6.8: Average found time with $Fp = 10\%$

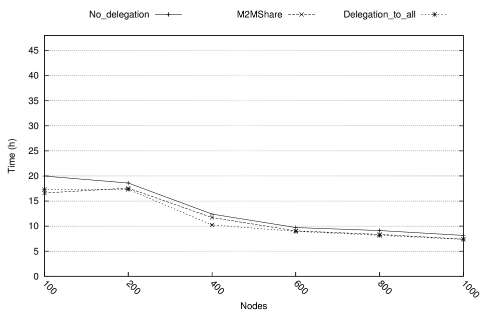


Figure 6.9: Average found time with $Fp = 50\%$

6.4 Multi-hop delegation

In previous analysis we considered only delegation strategies using single-hop delegation, i.e. once a servant peer receive a task, delegated from another node, it will not delegate it again to a further-lever servant.

There are some situations in which single-hop delegations are not enough, due to other factors, like a low popularity of the searched file or its high distance from the searching peer. It is also possible that all the peers holding the searched file have different behaviours from those of the client and his direct servants. In these cases, we extended M2MShare giving to a peer the capability to delegate again a pending task. To avoid creating an excessively large number of delegations, we allow to delegate again only after a trial period, i.e. one day, in which the servant try to complete the task by itself. If at the end of this period the task is still incomplete, it is delegate again to a new set of upper-level servants.

Population	1000
File size	3.0 MB
File popularity	25 copies in a single district
Delegation type	No_Delegation, M2MShare
Delegation depth	1, 3
File Division Strategy	M2MShare
Nr. of simulations	40 x 3
Simulated time	One week

We simulate the behaviour of our protocol in a similar situation by tuning the distribution of the searched file at the beginning of the simulation: we distribute the 25 initial copies only between nodes in a map district on the other side of town from the searching node. As usually we repeated the simulations several times to get more accurate results, independently from the initial location of the searching node. In every simulation we compare then the effectiveness of three different strategies:

- a protocol which don't use delegation
- M2MShare with 1-hop delegations
- M2MShare with up to 3-hop delegations

In Fig. 6.10 it is possible no see the average percent value of simulations where nodes employing the three compared strategies successfully found the searched file and bring it back to the requester node. M2MShare with 1-hop delegations can achieve some success, but it is not comparable with results of M2MShare employing multi-hop delegations. Finally, the strategy not employing delegations is not able to find the file a single time.

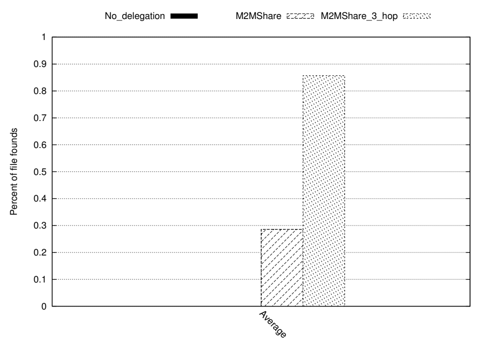


Figure 6.10: Percentage of successfully completed simulations.

It is straightforward that delegating again a unaccomplished task extends the total explored area. As it is possible to see in Fig. 6.11, the strategy which don't use delegations explores only a small area, covered only by its own connectivity range. With M2MShare (in Fig. 6.12) there is an increase of the explored area, due to delegations to some servants with different movement behaviour than the requester, but limited to 1-hop delegation. The maximum area extension is when using up to 3-hop delegations in which, as we can see in Fig. 6.13, almost all the city is covered by searching nodes.

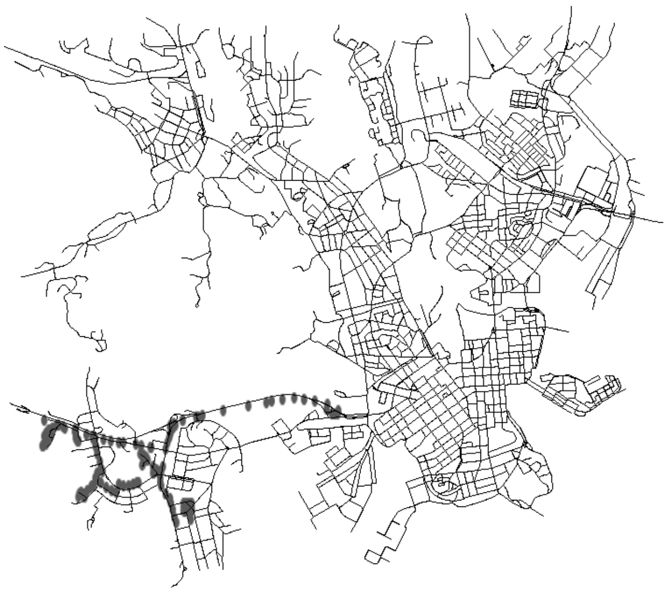


Figure 6.11: Explored area without use of delegations

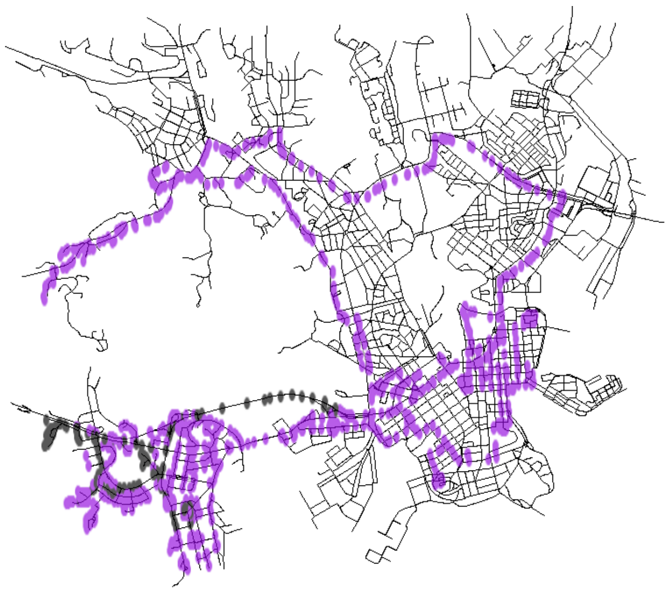


Figure 6.12: Explored area using M2MShare and 1-hop delegations

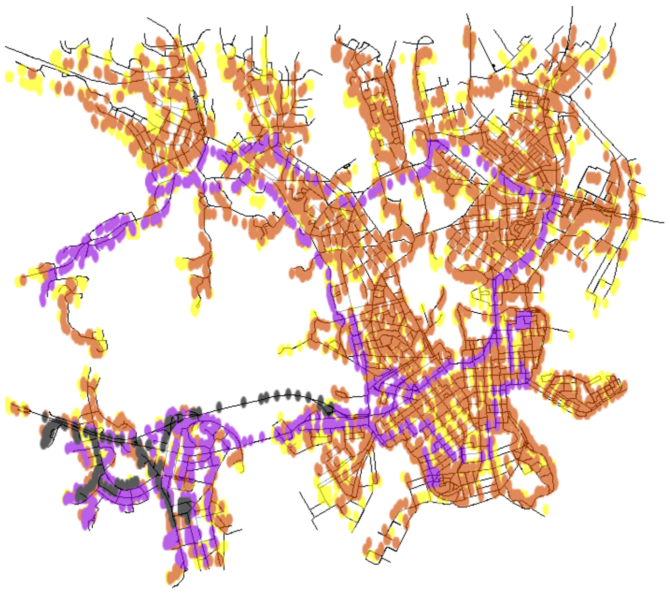


Figure 6.13: Explored area using M2MShare and up to 3-hop delegations

6.5 Data redundancy

Population	1000
File size	3.0 MB
File popularity	50 copies uniformly chosen
Delegation type	M2MShare, Delegation_to_all
Delegation depth	1
File Division Strategy	M2MShare
Nr. of simulations	40 x 2
Simulated time	One week

In analysis in Section 6.1, especially in Fig. 6.5, we show that our system is the most efficient with respect to data transmissions. Although using delegations introduces an overhead in terms of bandwidth usage we control this side effects by delegating only to frequently encountered peers whom are expected to be encountered again in the future. Another side effect caused by delegating tasks is the increasing of data redundancy in the whole network. For *redundancy*, in this case, we mean storage space used in nodes involved in delegation system.

Of course redundancy in a network composed only by nodes which don't use delegation is always zero. For this reason we compared for this study only the two systems which use task delegations. In Fig. 6.14 we show how change the average data redundancy during the progress of simulations. It is straightforward to notice the higher value introduced by delegating to all encountered peers whether M2MShare reduces the data redundancy quantity while still achieving the goal of acquiring the requested file. This is due to the number of contemporary active delegated tasks, shown in Fig. 6.15, which is higher in the system which delegates task to all encounter nodes.
The trend of this graph is related to the number of simultaneously active delegated tasks, shown in Fig. 6.15. Whenever a task is delegated, there is one more node looking for the searched file, and if it is found, the node will copy some file interval in its own data storage, doing so increase the total data redundancy. On the other hand, when a delegated task expires, or a servant returns the output to the requester node, temporary data downloaded for the task is deleted, freeing space in servant's data storage and doing decrease the total data redundancy.

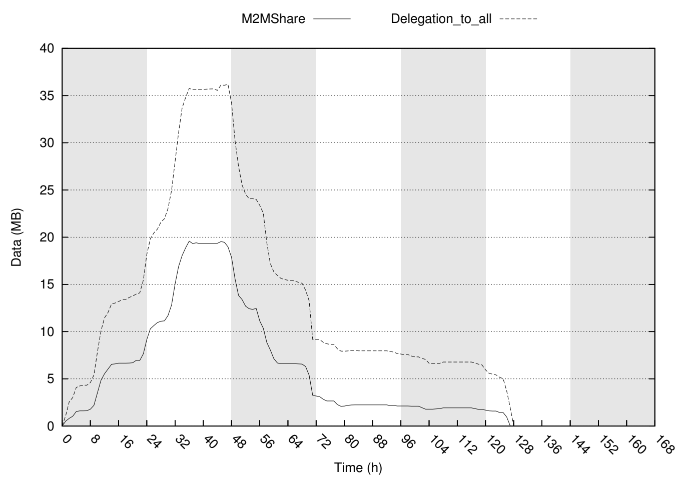


Figure 6.14: Average data redundancy in the network

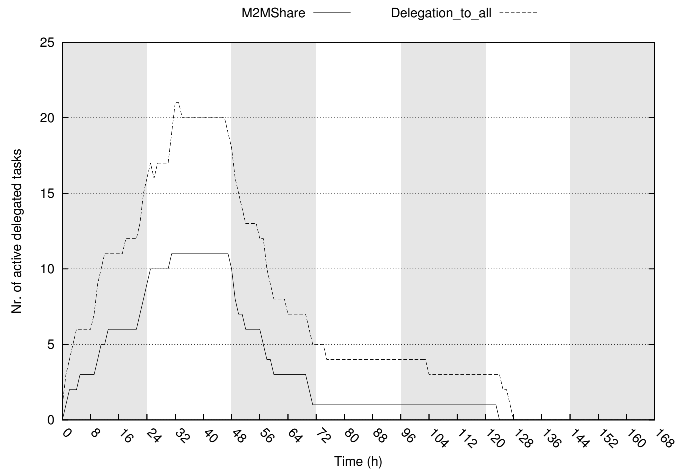


Figure 6.15: Average number of simultaneously active delegated tasks

6.6 File Division Strategy efficency

In previous analysis we evaluate the efficiency of M2MShare delegation technique, but in the protocol the delegation system is not the only new innovation described. M2MShare provides a new file division strategy, described in Section 4.6, where a file can be downloaded in pieces and a piece size varies. That file division strategy might add redundancy during data transfer as it can happen that overlapping data intervals are simultaneously downloaded by different servants. However, the fact that each servant is asked to download the file starting from different points allows reconstructing the whole file even if both downloaded just part of it.
In this analysis we compare our file division strategy with two other division strategies:

Population	1000
File size	3.0 MB, 10.0 MB, 25.0 MB
File popularity	50 copies uniformly chosen
Delegation type	M2MShare
Delegation depth	1
File Division Strategy	M2MShare, iM , rM
Nr. of simulations	40 x 3 x 3
Simulated time	One week

- iM: a strategy which requests at each file server the entire file, always starting from its first byte;
- rM: a strategy that randomly chooses the initial download point in the file request.

We considered the average, min and max transferred data amount employed in each file division strategy. We repeated the simulations 40 times, changing random seeds, for every file division strategy in order to achieve more accurate results and we did this using three different sizes of the searched file.
In earlier studies some test were done to evaluate the efficiency of M2MShare file division strategy, but task delegation was not considered because simulated scenarios did not take into consideration disconnections due to mobility or interferences that happen in real world communication. That has only been possible with the current implementation of the protocol, in a network simulation environment which can simulate movement of nodes emulating people using M2MShare. As we can see from the graphics below our division strategy has the least redundancy during data transfer, especially considering big-sized files.

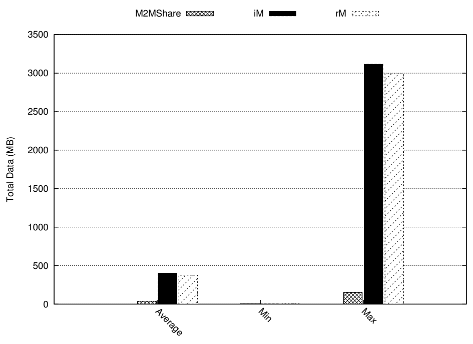


Figure 6.16: Average, min, max transferred data amount using different file division strategies and 3.0 MB file size.

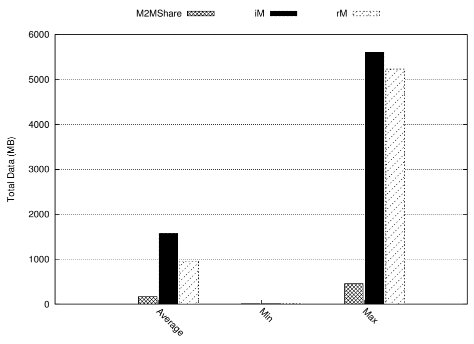


Figure 6.17: Average, min, max transferred data amount using different file division strategies and 10.0 MB file size.

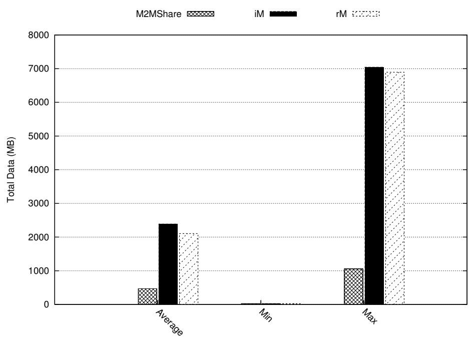


Figure 6.18: Average, min, max transferred data amount using different file division strategies and 25.0 MB file size.

Chapter 7

Conclusioni

LE conclusioni di tutto

Bibliography

[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.