Friendship-based Routing Protocol for Delay-Tolerant Networks

Francisco Fernandes, Naercio Magaia, Paulo Rogério Pereira
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisboa, Portugal
{fxameliofernandes, naercio.magaia}@tecnico.ulisboa.pt, prbp@inesc.pt

Abstract— **Delay-Tolerant Networks are characterized by not having permanent end-to-end connections, often leading to intermittent connectivity, long and variable delays and high error rates. Mobile Social Networks are particular scenarios on which nodes are seen as individuals with inherent social habits, having wireless communication devices. In this paper a routing protocol based on the social property of Friendship is proposed. The protocol allows nodes to consider other nodes to be friends if they maintain contact frequently, regularly and in long-lasting sessions. The forwarding scheme consists in only delivering the message to nodes which are friends of the destination. To evaluate the performance of this protocol, the ONE Simulator was used. The Friendship Protocol performance was analyzed and compared to three other routing protocols while varying the network load. The simulation results showed that the Friendship Protocol could reach a high delivery rate and a very low overhead. A dynamic threshold version of the protocol was also proposed, on which the friendship threshold changes over time as it corresponds to a portion of the best friends. The results of this dynamic version were similar to the fixed threshold version, but with roughly half the overhead.**

Keywords- Vehicular Delay-Tolerant Networks, Mobile Social Networks, Routing Protocol, Friendship.

I. INTRODUCTION

Vehicular Delay-Tolerant Networks (VDTNs) [1] are networks with a variable topology over time, in which there is no permanent path between the source and the destination for data routing. The main characteristics of VDTNs are: intermittent connectivity, long and variable delays, and high error rates in data transmission. Due to frequent network partitions, the usual routing protocols cannot find routes resulting in data transmission failures. Routing in VDTNs is done through cooperation between the vehicles (i.e., nodes) of the network. Messages are stored in the nodes' buffers and carried. When another node with better conditions to find the destination appears, the message is transferred, repeating the process until the destination of the message is eventually found.

Vehicles are driven by people that have social habits and some regular movement patterns, such as going from home to work, from work to home, shopping, meeting with friends, etc. These social habits may be learned by nodes in order to improve message routing.

In this work, the Friendship protocol [10], a Delay-Tolerant Network (DTN) routing protocol that takes into consideration the learned social relationship between nodes in the forwarding decision, is implemented in The Opportunistic Networking Environment (ONE) simulator [14]. The protocol is extended with a dynamic friendship threshold to avoid manual configuration and its performance is assessed through simulation in a VDTN.

The rest of the paper is organized as follows: in Section 2, an overview of previous work is presented. In Section 3, the details of the design of the Friendship protocol are given. In Section 4, the results of the performance simulation are analyzed and compared to other protocols. Finally, some conclusions are drawn in Section 5.

II. RELATED WORK

A. DTN Architecture Overview

To overcome the challenge of having no end-to-end path between source and destination during a communication session, the DTN architecture was specified [2]. This architecture is based on particular design principles such as the use of variable-length messages (not limited-sized packets) called "bundles" to improve the scheduling and path selection decisions and the use of storage within the network to support store-carry-and-forward operations over long timescales and multiple paths. It is possible that several copies of the same bundle may co-exist in the network, stored in the memory of different nodes.

B. DTN Applications

The popularity of DTNs is increasing as different practical usages are appearing on very different fields. There are several examples of challenging scenarios on which DTN research can directly be applied.

The so-called Mobile Social Networks (MSNs), is an increasingly popular type of DTNs. MSNs are of growing significance as a result of the explosive deployment of mobile personal wireless devices among people, such as in vehicles, smartphones and GPS devices. Such devices can generally transfer data in two ways [3]: by transmitting it over a wireless network interface or by being carried by its user from location to location. The established connections are therefore seen as "opportunities" that arise whenever mobile devices come into wireless range due to the mobility of their users.

C. VDTN Routing Protocols

VDTN routing presents the challenge of finding the most adequate node to forward messages to in the scenario that end-to-end paths might not exist at all time. For this reason, VDTN routing protocols employ a store-carry-and-forward approach, i.e. nodes hold messages until a suitable node to forward them is found.

Routing schemes are usually evaluated by some common metrics. In a general way, three basic metrics could be defined [4]: Delivery Ratio, the ratio between the number of delivered messages over the number of generated messages; Overhead Ratio, the ratio between the number of total transmissions over the total number of messages delivered; and Delivery Delay, which is the time duration between the messages generation and delivery.

The routing objective provides a tradeoff between minimizing the overhead ratio and maximizing the delivery ratio. Although VDTNs' applications are inherently tolerant to long delivery delays, lowering the delay is a target.

Based on [5], routing protocols can be summarized into two major categories, *social-oblivious* and *social-aware* (also known as traditional and social-based) based on the information that nodes take into account for making forwarding decisions. In social-oblivious protocols, a certain number of message replicas are diffused through the network in hope that one will eventually reach the destination. On the other hand, social-aware protocols significantly rely on the nodes' social relations to route messages to the most promising next hop in terms of probability of success of the delivery.

1) Social-oblivious family

Within VDTN routing protocols, the social-oblivious family relies on the replication approach to achieve sufficient delivery without considering the candidate node selection. These protocols are generally simple to implement as they do not require each node to have knowledge about the network. Epidemic [6] spreads the message through all nodes encountered and primes for maximum delivery ratio when usage of resources (e.g. buffer space and bandwidth) are not taken in account. The protocol is based on the process of flooding: each node that carries a message will replicate it to every neighbor available in its range if it does not have yet a replica. Research shows that delivery success ratio can be high but at the cost of a high buffer space used by each node and a high transmission overhead. The PRoPHET protocol [7] is a more sophisticated protocol that calculates delivery probabilities of nodes. The estimated delivery probability increases whenever there is a direct contact with a node or with a node that has a high probability of meeting the target node, and decreases with time if there are no encounters. If an encountered node has a higher delivery probability than the node that carries the message, a replica of that message will be sent to the encountered node.

2) Social-aware family

Social-aware protocols, in turn, explore the social behavior of the nodes that compose a VDTN. This kind of protocols not only deals with dynamic network information (e.g. instantaneous location and encounters) but also aims to explore social relations among nodes. These protocols exploit several social proprieties, also called social metrics that derive from Sociology. Surveys such as [4], [8] and [9] suggest that there are five main social proprieties explored by the majority of the proposed VDTN social protocols: Community, Friendship, Centrality, Similarity and Selfishness. Community refers to groups of people living in the same place or having a particular characteristic in common. Friendship describes close personal relationships, as it has been observed [9] that individuals generally establish friendships with others that share the same interests, perform similar actions and frequently meet. Centrality describes the social importance of a person in a social network. Similarity measures the degree of separation between individuals in social networks and presupposes that there is a higher probability of two people connecting if they have connections in common. Selfishness refers to nodes that maximize their own utility either by dropping others' messages or by excessively replicating their own.

The BubbleRap protocol [11] is based on two social proprieties, community and centrality. BubbleRap is based on the Label [12] and Rank [13] algorithms. The former uses explicit labels in nodes to identify the communities that they belong to, while the latter relays messages to nodes with higher centrality than the current node. The forwarding mechanism in BubbleRap works in a way that if a node wants to send a message to a certain destination, first it should "bubble" this message to a node which has higher global rank until the message reaches a node which has the same label of the destination node's community. After that, global ranking is ignored and instead messages are forwarded based on local ranks. In this way, messages will either successfully reach destinations or eventually expire. Contrary to what happened in the Rank algorithm, in BubbleRap nodes are able to consider global rankings even though they do not know the entire ranking table. Each node has the capability to detect the communities it belongs to and calculate its centrality values. Communities are detected by one of two possible ways, which are using labels or using a distributed mechanism.

Friendship Based Routing [10] is a single-copy protocol which, as its name suggests, adopts Friendship as its key factor. The community concept is also explored as each node has its own group of friends, which are regarded as close relationships on which forward opportunities are frequent. Communities are based on each node's point of view and utilize time dependent interactions with others, and would be the primary criteria for forwarding messages. The community is a set of nodes that are direct or indirect friends and therefore messages should be sent to a contact only if the destination is among their friends. Researchers in [10] found a link metric, designated as the *Social Pressure Metric* (SPM), that reflects the node's relations in a more accurate way than the previous existing metrics. It is considered that for two nodes to be friends, they need to contact frequently, regularly, and in long-lasting sessions. Indirect friends are also included in the community. This happens when nodes have a very close friend in common so that they can contact frequently through this common friend. In order to identify those indirect friendships, the *relative SPM metric* (RSPM) is proposed, which is a quantity that represents the average delivery delay of a message that followed the path

<*i,j,k*> if messages are generated at every time instant. Also, it is believed that communities should address the periodic variations of relationships. This concept leads to the determination of different communities for each time interval of the day, in order to depict nodes' routines to make better routing decisions.

III. FRIENDSHIP PROTOCOL

A. The Concept

The Friendship protocol is a social-based routing protocol for DTNs which utilizes the ideas proposed in [10]. It is a protocol meant to be used in a MSN scenario on which the nodes are human people with a daily life. Since the source code adopted by the original authors was unavailable, the interpretation and implementation of this protocol is our own and adjusted to take advantage of The ONE simulator [14] functionalities. This protocol takes into account that node relations often change with time periodically and addresses the fact that people's main activities often occur with regularity, so the friendship communities are periodic and take respect to a certain period, or *timeslot*, of the day. It was empirically determined that the duration of each timeslot would be 3 hours, being a 24 hour day a set of 8 timeslots, as it led to the best results.

B. The Friendship Protocol

Time is very important for the Friendship protocol. As previously stated, nodes analyze encounter information independently at each timeslot. Each node has access to the global time (in seconds) of the simulator, as if each one had a clock that started counting on the exact time the simulation started. This global time counting is crucial to measure time events. However, time must be manipulated properly to allow nodes to situate themselves in terms of timeslots. Nodes must be aware of which timeslot they are in order to assign friendship communities to the respective timeslot. Algorithm 1 is an algorithm which takes as an input a global time *t* in seconds and uses it to obtain some useful information about the current timeslot, namely the current timeslot index, the start and end instant of the corresponding timeslot and on which day of simulation corresponds the global time.

Algorithm 1 – *getTimeSlotInformation*

Input: Time *t*;
Output: Timeslot *timeslot_index*; Time *start_of_timeslot*; Time end_of_timeslot; Day *sim_day;*

1: *sim_day* = ⌈ *t* / 86400 ⌉
2: **if** *t* is a multiple of 86400 **then**
3: sim_day++
4:**end if**
5: start_of_timeslot = (*sim_day*-1)*86400
6: end_of_timeslot = *start_of_timeslot* + 3 hours
7: **for** *timeslot_index* from 1 to 8
8: **if** time *t* is between the start and end of the timeslot **then**
9: **return** *timeslot_index, start_of_timeslot, end_of_timeslot* and *sim_day*
10: **else**
11: *start_of_timeslot* += 3 hours
12: *end_of_timeslot* += 3 hours
13: **end if**

Algorithm 2 has a key role in this protocol as it converts global time to local time, regarding the timeslot. In essence, this algorithm "corrects" the global time values by defining a new origin of the time axis that corresponds to the beginning of the timeslot in matter.

Algorithm 2 – *timeCorrect*

Input: Time *t*;
Output: Time *t_corrected*
1: *start_of_timeslot, sim_day = getTimeSlotInformation(t);*
2: *t_corrected*=(*sim_day*-1)* 3 hours + (*t* - *start_of_timeslot)*
3: **return** *t_corrected*

The next two algorithms, Algorithm 3 and 4, are part of the routines which are called whenever a connection is established or lost, respectively. The main purpose of these algorithms is to update the SPM and RSPM metrics. The SPM update process will be explained firstly. The SPM definition from node *i* to *j* is

$$SPM_{i,j} = \frac{\int_{t=0}^{T} f(t)dt}{T}, \quad (1)$$

where $f(t)$ represents the time remaining to the next encounter of the two nodes at time *t*. It is possible to simplify the expression to facilitate its computation by each node. If there are *n* intermeeting times in time period *T*, then the SPM from node *i* to *j* is

$$SPM_{i,j} = \frac{\sum_{x=1}^{n} t_{inter,x}^2}{2T}, \quad (2)$$

being $t_{inter,x}$ the x^{th} intermeeting time value registered. In this way, nodes do not have to keep a record of the entire encounter history with each node, which would be expensive in terms of memory and CPU, only needing to keep summing to the numerator squares of the intermeeting times and dividing it by the double of the current time.

The process of updating RSPM also extends to both Algorithms 3 and 4. The RSPM represents the average delivery delay towards a certain node if the two-hop path <i,j,k> is taken. Two stages are considered. The first one, stage *a*, starts at the last meeting of node *i* with node *j* and ends at the time node *i*'s next contact with *j* ends, assuming that any message generated at node *i* can be transferred to node *j* when they are in contact. However, if there are any subsequent meetings with *j* before any meeting of *j* with *k*, then the last one is considered. During this stage, node *i* transfers messages to node *j*. The duration of this stage is $t_{a,x}$. The second stage, stage *b,* starts when the first one ends and finishes when node *j* meets *k*. The duration of this stage is $t_{b,x}$. Denoting the total number of sessions as *n*, the definition of RSPM from node *i* to *k* crossing *j* is

$$RSPM_{i,k|j} = \frac{1}{T} \times \sum_{x=1}^{n} \int_{t=0}^{t_{a,x}} (t_{a,x} + t_{b,x} - t)dt \quad (3)$$

In order to ease the computation process at each node, this expression can be simplified to

$$RSPM_{i,k|j} = \frac{\sum_{x=1}^{n} 2t_{b,x}t_{a,x} + t_{a,x}^2}{2T}. \quad (4)$$

The node responsible for the computation of the RSPMs is the intermediate node *j*, which has direct access to both records, so every node should maintain updated the RSPM values on which it is the intermediate node. This node is afterwards responsible of informing other nodes about their RSPM values on which the intermediate node is itself.

Algorithm 3 – *neighborDetected*

Input: Time *t*; Node *otherHost*;
Output: (none);
1: *timeslot_index, start_of_timeslot, end_of_timeslot, sim_day = getTimeSlotInformation(t)*
2: *time_corrected=timeCorrect(t)*
3: *start_of_encounter(timeslot_index,otherHost = time_corrected*
4: **if** there are no records of encountering *otherHost* in timeslot *timeslot_index* **then**
5: **initialize** *end_of_encounter(timeslot_index,otherHost)* = 0 seconds *totalS(timeslot_index,otherHost)* = 0 seconds^2
6: **end if**
7: *intermeeting_time= start_of_encounter(timeslot_index,otherHost) - end_of_encounter(timeslot_index,otherHost)*
8: *previous_totalS = totalS(timeslot_index,otherHost)*
9: *totalS(timeslot_index,otherHost) = previous_totalS + (intermeeting_time)^2*
10: *SPM(timeslot_index,otherHost)= totalS(timeslot_index,otherHost) / (2*time_corrected)*
11: **for** every other host *k* in the network besides the encountered node
12: endB*(timeslot_index,otherHost,k)=time_corrected*
13: **if** there are no records of encountering *otherHost* in timeslot *timeslot_index* **then**
14: **initialize** *startA(timeslot_index,otherHost,k)= 0 seconds* **and** *endA(timeslot_index,otherHost,k)= 0 seconds* **and** *totalR(timeslot_index,otherHost,k)=0 seconds^2*
16: **end if**
17: **if** *endA(timeslot_index, k, otherHost)* **is superior than** *startA(timeslot_index, k, otherHost)* ***then***
18: *tb* = endB*(timeslot_index, k, otherHost)- startB(timeslot_index, k, otherHost)*
19: *ta* = endA*(timeslot_index, k, otherHost)- startA(timeslot_index, k, otherHost)*
20: *previous_totalR = totalR(timeslot_index, k, otherHost)*
21: *totalR(timeslot_index, k, otherHost)= previous_totalR * 2* ta * tb + ta^2*
22: *RSPM(timeslot_index, k, otherHost)= totalR(timeslot_index, k, otherHost) / (2*time_corrected)*
23: *startA(timeslot_index, k, otherHost) = endA(timeslot_index, k, otherHost)*
24: **end if**
25: **end for**
26: **return**

Algorithm 4 – *neighborLeft*

Input: Time *t*; Node *otherHost*;
Output: (none);
1: *timeslot_index, start_of_timeslot, end_of_timeslot, sim_day = getTimeSlotInformation(t)*
2: *time_corrected=timeCorrect(t)*
3: **if** a timeslot overlap occurred **then**
4: *neighborLeft(otherHost, start_of_timeslot-0.1)*
5: *neighborDetected(otherHost, start_of_timeslot)*
6: **else**
7: *end_of_encounter(timeslot_index,otherHost) = time_corrected*
8: *SPM(timeslot_index,otherHost)= totalS(timeslot_index,otherHost) / (2*time_corrected)*
9: **for** every other host *k* in the network besides the node that has just left
10: *endA(timeslot_index,otherHost,k)= time_corrected*
11: *startB(timeslot_index,otherHost,k)= time_corrected*
12: **end for**
13: **return**

The next two algorithms, Algorithm 5 and 6, represent the two situations where decisions are made based on friendship weights. A link weight ω, hereby designated as friendship weight since it describes in essence how strong a friendship is, is defined as the inverse quantity of SPM or RSPM, whichever the lowest. In this way, the lower the SPM / RSPM quantity, the higher is the friendship weight. τ is the minimum friendship weight for a node be considered as a friend and thus part of the community of friends, which always concern a certain timeslot, resulting in nodes having 8 different communities corresponding to the 8 timeslots of the day. One shall note that, for indirect friends, it is required that the intermediate node is a friend itself in order to add them to the friend network (FN). Having this fact in mind, it would only make sense to ask RSPM information from an encountered node if that node itself is part of our community. Algorithm 5 shows the procedure for requesting RSPM values upon an encounter.

Algorithm 5 – *requestRSPM*

Input: Time *t*; Node *otherHost*
Output: Boolean
1: *timeslot_index = getTimeSlotInformation(t);*
2: **if** the friendship weight ω towards *otherHost* is above the threshold τ on the current timeslot **then**
3: **request** RSPM values regarding itself towards every other node on which *otherHost* is the intermediate node
4: **return** true
5: **else**
6: **return** false

Algorithm 6 is the last algorithm to be executed upon a new connection being established and it is when the node actually decides if it is going to forward or not the messages stored in its buffer. At this point, Algorithm 3 and 5 have been run, so the node has its community updated in the current timeslot. Algorithm 6 will be called for each message stored in buffer. In the case that the encountered node is the final destination of the

message (line 2), the message is forwarded and deleted from the buffer. If the destination of the message is part of the FN of the encountered node (line 7), then the message is sent and deleted from the buffer only if their friendship is stronger (line 8). In every other case, the message remains with the host and is not sent. In this way, messages are only relayed to nodes which are friends of the destination and their friendship is stronger than ours towards the final recipient.

Algorithm 6 – *shouldSendMessageToHost*

Input: Time *t*; Message *m*; Node *otherHost*;
Output: Boolean "answer";
1: *timeslot_index = getTimeSlotInformation(t);*
2: **if** the *otherHost* is the destination of the message *m* **then**
4: send message *m* to *otherHost* and delete *m* from buffer
5: **return** true
6: **end if**
7: **if** the destination of the message is on *otherHost*'s friendship community **then**
8: **if** *otherHost's* friendship weight ω towards destination **is superior** to ours **then**
9: send message *m* to *otherHost* and delete *m* from buffer
10: **return** true
11: **end if**
12: **return** false
13: **end if**
14: **return** false

C. A dynamic threshold version

The protocol presented in the previous section was implemented respecting the idea that the threshold τ for a friendship should be a pre-defined value that every node should adopt as the minimum required to be part of their respective community of friends. However, a fixed value may not be adequate for all scenarios and the best value can be difficult to determine.

A dynamic version of the Friendship's threshold is proposed in this paper on which the threshold to use is independently determined by each node. It may be beneficial to assume that since nodes have different routines and encounter frequencies, they should also have different friendship perceptions. The idea is that the friendship threshold should not be a fixed value that each node should adopt, but instead be a portion of its best friends. By comparing every friendship with its best friendship, nodes could adjust the number of nodes in their communities based in the amount of activity that each one has. Because the friendship weights of the nodes' friends change over time, the threshold would vary along, resulting in a dynamism that would follow each node's vision in terms of friends. To differentiate each version of the Friendship Protocol, the one proposed in this section is hereby designated as Dynamic Friendship while the other is designated as the Classic Friendship version.

IV. Analysis of Results

The protocols chosen to be tested along with Friendship are Epidemic, Prophet and BubbleRap protocols, which all have particular characteristics that might be drawbacks when compared with Friendship. The protocols were evaluated using The ONE Simulator [14]. In Phase 1, the primary objective is to tune BubbleRap's parameters number of k-cliques (*k*) and *Familiar Threshold* [11] and Friendship's *Friendship Threshold* in order to optimize their performance under the given scenario conditions. The Familiar Threshold corresponds in BubbleRap to the minimum time which a node must, in aggregate, remain in contact with another in order to be considered part of its community, while the Friendship Threshold in the Friendship Protocol is the minimum friendship weight a node must achieve to be considered a friend. Phase 2 has the purpose of assessing the effect of traffic load in the network by varying the number of messages created per time unit, now that we have the protocols optimized. It should be interesting to see how the protocols' performance depends on the network load as it should give us a wider picture of their behavior. Furthermore, it should help us clarify the consequences of having a dynamic tuning mechanism in detriment of a fixed threshold.

A. Phase 1: Tuning

The simulation parameters were chosen with the intention of having a realistic scenario and remained unchanged on every protocol test.

A scenario was built with three types of nodes circulating on a map of Helsinki, for a total of 114 nodes: pedestrians (90), trams (12) and buses (12). The main idea is that both pedestrians and vehicles possess a wireless device that they use to exchange messages with others. Trams and buses have the Routed Map-Based Movement (RMBM) pattern, which means that both groups follow a regular pattern between two location points over time, with brief stops on the way. The main difference between these vehicles is that pedestrians are only allowed to ride buses, as if they had only a bus pass and not a tram pass. Buses and trams that are within the same group follow the same movement pattern. Pedestrians, in turn, follow the much more complex Working Day Movement Model (WDM) pattern. These nodes are programmed to have a daily routine resembling a working individual which has a home, a job office and favorite socializing spots. Whereas the home and job office of each pedestrian is randomly located at any point of the map, pedestrians of the same group are limited to a certain zone of the city to socialize. Also, pedestrians within the same group can only ride the buses that are located near their favorite meeting spots and no other vehicles. All pedestrians spend about 8 hours at work and after that they have a 75% probability to go socializing for 1 to 2 hours with 2 to 8 other people. They always return home until the next morning, and then repeat their routine.

The buffer size represents the space available at a node to store messages. It was set at 1 GB. It is important to refer also that every time a message arrives at a node and there is no buffer space available, a default drop policy is triggered where the node will search for the oldest messages in the buffer and remove them until there is enough space for the incoming one.

In terms of network interfaces, all groups of nodes use a single Wi-Fi interface with a constant throughput of 10 Mbit/s and limited to a communication range of 10 meters. This is a

simplified model of the simulator configured to take into account average outdoor conditions.

Message sizes vary randomly from 3 MB to 15 MB and are created at a rate of 3 messages per minute. It was also established that each message would have a maximum of 1.5 days to live (time-to-live or TTL).

In order to gather more consistent results, four simulations with different seeds were run in every test. Therefore, each value presented is an average of the results for these four simulations with a confidence interval associated. In all cases, a 95% confidence interval is considered. The simulation time is 8 days on all cases.

The performance chosen as the best for the BubbleRap protocol occurred for a *familiar threshold* of 2500 and a *k* of 13, resulting on a delivery rate of 30.6%, an overhead ratio of 41.8 and an average delay of 14258 seconds, which approximately corresponds to 4 hours. In the case of the classic Friendship protocol, although it was hard to find a conclusion about which value led to the optimal overall performance since the results did not differ much in terms of delivery rate according to the confidence interval, the performance chosen as optimal was for the *Social Pressure* (SP) threshold, which is merely the inverse of the weight ($1/\tau$), set to 2 hours and 15 minutes, resulting in a maximum delivery rate of 36.5%, an overhead ratio of 2.4 and an average delay of 18858 seconds (5.24 hours). For the dynamic version of Friendship, the dynamic threshold chosen as optimal was 80% of the best friendship, which resulted in a delivery rate of 35.2%, an overhead ratio of 1.13 and an average latency of 18378 seconds (approximately 5.1 hours).

B. *Phase 2: Traffic load variation*

The next task was to vary the network load and assess each protocol's behavior under different congestion conditions. The scenario used was exactly the same as the previous, except for the message generation intervals parameter. As the purpose of this phase is to determine how the protocols' performance depends on the network load, the simulations were now run for ten different message generation intervals: 6, 7, 8, 10, 12, 15, 20, 30, 60 and 120 seconds.

The results of the evolution of the delivery rate metric with the message generation rate in the network are depicted in Figure 1. Both versions of the Friendship Protocol had good performance in terms of the delivery rate metric, as no other protocol tested showed better results according to the confidence interval.

Regarding the overhead ratio metric, the results over the different message generation rates are shown in Figure 2 in a logarithmical scale. Friendship is a social propriety which describes close personal relationships, so it was expected beforehand that message relaying would be much more selective as messages only get forwarded to friends of the destination to avoid unnecessary transmissions. This revealed to be true, as the overhead ratio was substantially lower for Friendship than for all other protocols tested.

These results reveal the major benefit of using the dynamic version of Friendship in detriment of the classic version, as the overhead ratio of the dynamic version is roughly half of the classic's for all rates and thus reveals to be more efficient.

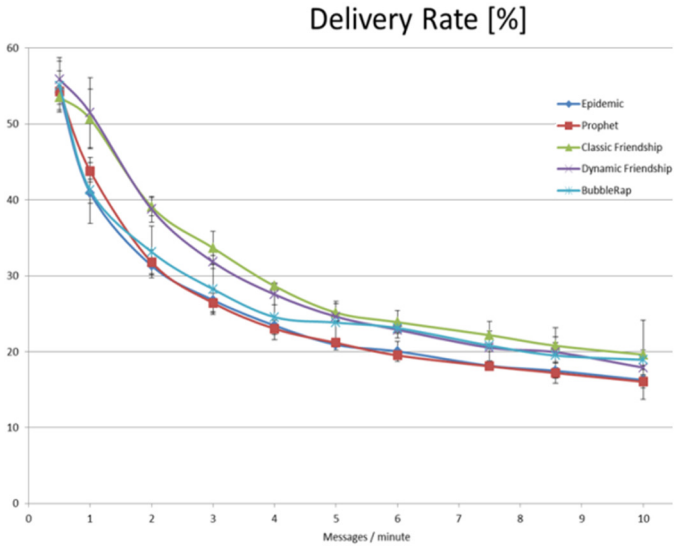

Figure 1. Delivery rate vs. message generation rate

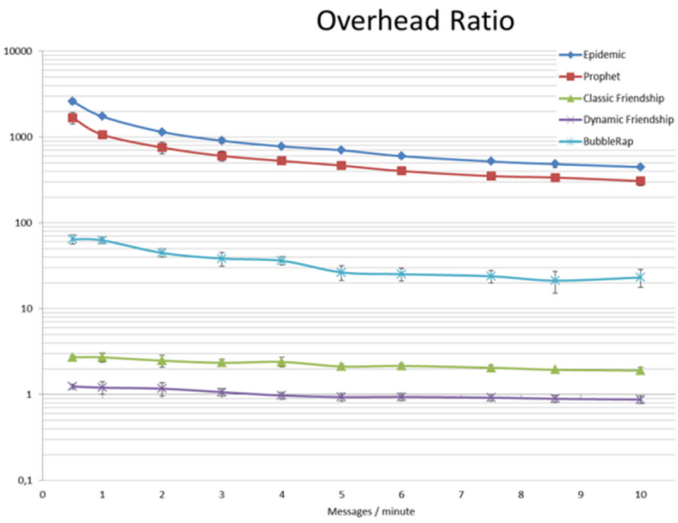

Figure 2. Overhead ratio vs. message generation rate

The results regarding the average delay metric are presented in Figure 3. The average message latency decreases for every protocol tested when the message generation rate increases, since a significant part of the messages is discarded due to network congestion as several nodes have fully occupied buffers and have to discard old messages in order to receive new ones. These results expose the implications of using Friendship as the primary social property to relay messages, as it may take more time to find a friend of a destination and thus the average latency is increased, which is more evident in the case on which friendship is considered to be dynamic and specific to each node.

V. Conclusion

The main objective of this work was to assess and compare the performance of the Friendship Protocol, a social-aware

DTN routing protocol mainly based in the social concept of Friendship.

The Friendship protocol is based in two important concepts for social-aware routing protocols in DTNs: Friendship and Community. Node relations are always analyzed for a certain period of the day and are depicted through a friendship weight, which is the inverse of one of two metrics, which are SPM and RSPM. SPM indicates the average delay for meeting a certain node and RSPM the average delay that messages would suffer if they had to cross a certain intermediate node to reach another node. Nodes are considered to be friends if their friendship weight is superior to a certain threshold. Only nodes that are friends belong to their communities, and messages are only relayed if the destination is friend of the encountered node. As an alternative of a fixed threshold approach, a dynamic version of the protocol on which the threshold is a portion of the current best friendship is proposed, meaning that friendship standards change from node to node dynamically.

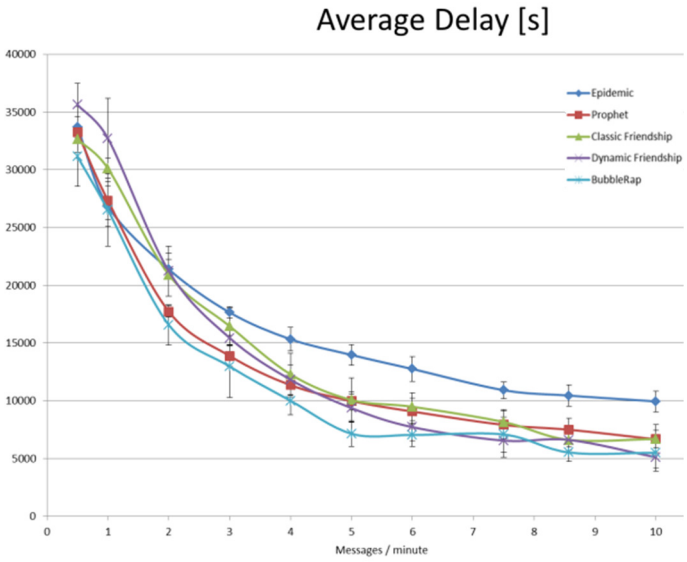

Figure 3. Average Delay vs. message generation rate

The protocol performance was compared to Epidemic's, Prophet's and BubbleRap's. The first phase of simulations was merely to adjust the configurable parameters of BubbleRap and Friendship. After the optimal parameters were found, the second phase was an assessment of how the protocols' performance changed with the number of messages in the network.

In terms of delivery rate, both versions of the Friendship Protocol had the best performance, for every network load value tested. The overhead ratio, in turn, was substantially lower for the two Friendship versions than for all other protocols tested as a single copy of each message circulates in the network. The results also revealed that the overhead ratio of the dynamic version is roughly half of the classic's. In terms of average delay, however, the two versions of Friendship led to worst values, as they generally took longer paths to get messages delivered when compared to other protocols.

The dynamic version of Friendship presents two main advantages when compared with the classic version: i) the overhead ratio of the dynamic version is roughly half of the classic's for all load values and thus reveals to be more efficient; ii) the friendship threshold does not need to be manually configured, as it is dynamically configured.

For future work, it would be interesting to collect some social data regarding BubbleRap's and Friendship's performance in the simulations, such as the number of communities formed or average number of nodes per community for BubbleRap or the average number of friends per node for Friendship, as that would definitely allow a more precise performance tuning. Additionally, it would also be relevant to test the protocols in different scenarios to check how Friendship behaves under conditions different than a MSN, the type of network which clearly it was designed for. Lastly, it could also be worthwhile to create a limited multiple-copy version of the Friendship Protocol, as that may result in significant performance gains in terms of delivery rate and delay at the expense of a small overhead increase.

ACKNOWLEDGMENT

This work was partially supported by Fundação Calouste Gulbenkian and by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

[1] P. Pereira, A. Casaca, J. Rodrigues, V. Soares, J. Triay, C. Cervelló-Pastor, "From Delay-Tolerant Networks to Vehicular Delay-Tolerant Networks", IEEE Communications Surveys & Tutorials, vol. 14, no. 4, pp. 1166–1182, 2012.

[2] V. Cerf, et al., *Delay-Tolerant Networking Architecture*, IETF RFC 4838, April 2007.

[3] Hui P., Chaintreau A., Scott J., Gass R., Crowcroft J., Diot C., *Pocket switched networks and human mobility in conference environments*, *Proceedings of ACM SIGCOMM workshop on DTN (WDTN)*, 2005.

[4] Cao and Sun, *Routing in Delay/Disruption Tolerant Networks: A Taxonomy, Survey and Challenges*, IEEE Communications Surveys & Tutorials, vol. 15, no. 2, pp. 654-677, 2013.

[5] C. Boldrini, K. Lee, M. Önen, J. Ott, E. Pagani, *Opportunistic networks*, Computer Communications 48, pp.1-4, 2014.

[6] A. Vahdat and D. Becker, *Epidemic routing for partially-connected ad hoc wireless networks*, Duke University Technical Report Cs-2000-06, Tech. Rep., 2000.

[7] A. Lindgren, A. Doria, E. Davies, and S. Grasic, *Probabilistic Routing Protocol for Intermittently Connected Networks*, IETF *RFC 6693*, 2012.

[8] F. Xia, L. Liu, J. Li, J. Ma, and A. V Vasilakos, *Socially-Aware Networking : A Survey* IEEE Syst. J., vol. 9, no. 3, pp. 1–18, 2015.

[9] Y. Zhu, B. Xu, X. Shi, and Y. Wang, *A survey of social-based routing in delay tolerant networks: Positive and negative social effects*, IEEE Communication Surveys & Tutorials, vol. 15, no. 1, pp. 387–401, 2013.

[10] E. Bulut, B. K. Szymanski, *Exploiting friendship relations for efficient routing in mobile social networks*, IEEE Trans. Parallel Distrib. Syst., vol. 23, no. 12, pp. 2254-2265, Dec. 2012.

[11] P. Hui, J. Crowcroft, and E. Yoneki, *BUBBLE Rap: Social-based forwarding in delay-tolerant networks*, *IEEE Trans. Mob. Comput.*, vol. 10, no. 11, pp. 1576–1589, 2011.

[12] P. Hui and J. Crowcroft, *How Small Labels create Big Improvements,* Workshops'07, 2007, no. April, pp. 65–70.

[13] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman, *Search in power-law networks*, Phys. Rev. E, vol. 64, 2001.

[14] A. Keränen, J. Ott, and T. Kärkkäinen, *The ONE Simulator for DTN Protocol Evaluation*, Proceedings of the Second International ICST Conference on Simulation Tools and Techniques, 2009.