Friendship-based Routing Protocol for Delay Tolerant Networks

F. X. Fernandes, MSc. Student, IST, Lisboa, Portugal

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. Sending a message between two points may take hours, weeks or even months. Mobile Social Networks are particular scenarios on which nodes are seen as individuals, with inherent social habits, carrying hand-held devices which communicate with each other within a certain wireless range. The scope of this paper was to develop a routing protocol based on the social property of Friendship to use on such networks, designated as Friendship Protocol. 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 analysed and compared to other three routing protocols while varying the network load. In the end, the Friendship Protocol showed that it could reach a high delivery rate and a very low overhead. For every network load tested, the results for the delivery rate were never lower than the other routing protocols and the overhead ratio was several times lower. It was also proposed a dynamic threshold version of the Friendship Protocol, on which the friendship threshold changes over time as it corresponds to a portion of the best friend weight. The results of this dynamic version were similar to the first version’s, only this time with roughly half of the overhead.

Index Terms—Delay Tolerant Networks, Mobile Social Networks, Routing Protocol, Friendship, The ONE Simulator.

I. INTRODUCTION

The motivation for research in Delay-tolerant Networks (DTNs) surged in the middle of the twentieth century with the competition between the Soviet Union and the United States for supremacy in spaceflight capability. As a response to the successful soviet campaign of launching a satellite to space, the United States created the Advanced Research Projects Agency (ARPA), which is known today as Defense Advanced Research Projects Agency (DARPA). After some time, this governmental agency started to fund multiple companies, like the National Aeronautics and Space Administration (NASA), in order to develop an Interplanetary Internet (IPN). As the architecture for an IPN started to be developed, it was detected early on that the existing technology needed to be adapted in order to support the obstacles of space, namely the high propagation delays, frequent disconnections and packet corruptions. Researchers rapidly understood that the protocols and algorithms used for terrestrial communications could not be the same ones used for space communications. After multiple research articles and conferences, there are nowadays several organizations focused in developing the DTN research area. Some of them are still focused on space communications but others already started working with other scenarios where the communications infrastructures are limited or inexistent and end-to-end connectivity may not always be available, which is the case of some developing countries, natural catastrophes or terrorist attacks.

This paper is motivated by the vision of how determinant can DTNs be in real scenarios of the present day world, where new networking possibilities were introduced by the extensive deployment of wireless devices. The work aims to take conclusions whether or not the Friendship protocol, a DTN routing protocol that takes into consideration the social relationship between nodes in the forwarding decision, is suitable for a real case scenario on which the nodes are people and try to communicate wirelessly without an existing infrastructure. Several tests were conducted throughout a simulator in order to assess the performance of the protocol.

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 analysed and compared to other protocols. Finally, some conclusions are offered in Section 5.

II. RELATED WORK

The characteristics of the environment can affect significantly the performance of a telecommunications network. Traditional end-to-end based routing algorithms are highly efficient when there is a well determined complete path from a source to a destination (e.g. the Internet). However, when the paths are unstable and suffer from constant disruption, these solutions work poorly and compromise end-to-end communications. In order to address the routing problem that arises from such challenged environments, a
different kind of networks with alternative routing algorithms was designed. These are called DTNs.

Routing schemes are usually evaluated by some common metrics. In a general way, three basic metrics could be defined [1]: 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 DTNs’ applications are inherently tolerant to long delivery delays, lowering this parameter should also be a target.

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 in RFC4838 [2]. The architecture is based on particular design principles such as the use of variable-length messages (not limited-sized packets) 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.

DTN services are provided by the Bundle Protocol (BP). The architecture works as a store-carry-and-forward overlay network. The BP introduces a new layer on the TCP/IP Stack Model called the “bundle layer”, between transport and application layers. The BP interfaces with different transport protocols through a “convergence layer” adapter. The convergence layer manages the protocol-specific details of interfacing with the underlying protocols and presents a consistent interface to the bundle layer.

The protocol data units are called “Bundles” and comprise all application layer data required for an interaction between end systems. These bundles are passed between entities that participate on BP communications, referred to as “bundle nodes”. A Bundle payload is the application data which is the purpose for the transmission (the actual message). It is possible that several instances of the same bundle may co-exist in the network, stored in the memory of multiple nodes.

B. 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 research on this field can directly be applied.

One application is deep space communications. The main purpose is to have inter-planetary communications over large distances and connect dispersed networks in space. The Multi-Purpose-End-To-End Robotic Operation Network (METERON) [13] is an application of DTNs for space communication on which rovers, a type of cars used for planetary explorations, can be controlled by astronauts from distance.

Another application of DTNs is wildlife tracking. A well-known project within the DTN research is the ZebraNet [14], which started in 2004. The purpose of this project is to permit biologists to obtain information about the mobility patterns of zebras from Kenya through a system composed by tracking collars and base stations.

As an application, there is also the so-called Mobile Social Networks (MSNs), 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 smartphones and GPS devices. Such devices can generally transfer data in two ways [19] – 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. Routing Protocols

DTN 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, DTN routing protocols employ a store-carry-and-forward approach, i.e. nodes hold messages until a suitable node to forward them is found.

The routing process can be unicast, multicast or anycast. Unicast routing relies on the principle that the destination is unique, while multicast routing implies that the destination is a group of nodes. Anycast routing, in turn, is a method on which the destination is any node in a group of potential receivers. Multicast and anycast routing fall outside the scope of this work.

Based on [3], 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 in account for taking 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 DTN routing protocols, the social-oblivious family relies on the replication approach to achieve a sufficient delivery without considering the candidate node selection. These are generally simple to implement as it does not require each node to have knowledge about the network. Direct Delivery [4] is a very simple protocol in which the source node constantly keeps the message until the destination is in proximity, not considering relay nodes. The number of hops required for delivery is only one, rather than multiple times using intermediate node forwarding, which may work when
the message Time-to-live (TTL) is long enough. However, it should be noted that a short TTL scenario may impose that messages never reach their destination due to long message delay. Epidemic [5] spreads the message through all nodes encountered and prunes 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 diffusion: each node that carries a message will replicate it to every neighbor available in its range if they do not have a replica. Research shows that delivery success ratio is high but at the cost of a high buffer space used by each node and a high transmission overhead. The ProPHET protocol [6] 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 DTN. This kind of protocols not only deal with dynamic network information (e.g. instantaneous location and encounters) but also aim to explore social relations among nodes. These protocols exploit several social proprieties, also called social metrics that derive from Sociology. Surveys such as [1], [7] and [8] suggest that there are five main social proprieties explored by the majority of the proposed DTN 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 [8] 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.

BubbleRap protocol [10] is based on two social proprieties, community and centrality. BubbleRap is based on the Label [11] and Rank [12] algorithms. The former uses explicit labels in nodes to identify the communities that they belong, 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, in the first place 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’s node 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 [9] 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 on [9] 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 on 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.

D. The ONE Simulator

It is possible to find in literature several simulators for routing protocols, such as the OMNeT++ [15], the ns Simulator, both versions ns-2 [16] and ns-3 [17], and The Opportunistic Networking Environment (ONE) Simulator [18]. These are all discrete event open-source simulators, which is ideal for research. The ONE Simulator was chosen to be used in this work due to the fact that it the only one originally designed just for the purpose of DTNs.

The ONE simulator is a Java agent-based discrete event simulator. The agents are called nodes, following the same nomenclature of the previous sections, and they are capable of simulating the store-carry-and-forward behavior. Each node possesses a set of capabilities as a radio interface, memory storage, movement and a certain energy consumption, which can be configurable. The simulator comes with a pre-defined number of modules, which are completely modifiable. The creator started by doing the core and some functionalities, and user contributions eventually appeared with time making it a community open project.

The most relevant features of this simulator are the modeling of node movement, inter-node contacts, routing and
message handling. Figure 1 shows a schematic of The ONE Simulator architecture.

Movement models consist in a set of algorithms and rules that are used to define the mobility pattern of nodes. To start, there are two implementations of random models, Random Walk (RW) and Random Waypoint (RWP). Movement in RW is random in a way that for each node, a destination is assigned, only for it to choose its own direction and speed from a predefined range. When the node reaches its destination, a new destination is assigned and so on. RWP works identically, the difference being that it assigns a random pause time on top of RW and direction, speed and pause assignment follow a uniform distribution. The Map-Based Movement (MBM) model is based on RW as nodes move in a random way on pre-defined paths on the map. When a path is selected, they follow it only to turn back when the end is reached or turn when an intersection of paths is found. The Routed Map-Based Movement (RMBM) model is based on the MBM and introduces pre-defined routes which nodes may follow. The Shortest Path Map-Based Movement (SPMBM) model, in turn, is another enhanced version of the MBM on which Points of Interest (POIs) are introduced. The POIs are points in the map which represent common places in a city, like shopping centers, schools or cinemas and so on. Each POI has a probability to be chosen different from other destinations that are not considered as a POI. Finally, in terms of Human-based models, there is one already implemented on The ONE, the Working Day Movement Model (WDM). As the name suggests, this model aims to replicate a typical day in the human society. To do that, this model considers three phases during a working day, which are sleeping, working and hanging out with after work, each of these with a different model. It also introduces the possibility of a node to travel alone or in groups and can have into account communities and social ties. The WDM is the most realistic model between the ones stated, as it simulates a typical MSN scenario.

Event Generators indicate how messages are generated in the simulator. When the simulator generates messages, it is possible to choose source, destination, size of messages and also the interval between messages.

Routing is the way messages are forwarded until they reach the destination. By default, the simulator comes with a number of social-oblivious protocols already implemented, which are Epidemic, Spray and Wait, PRoPHET, Direct Delivery, First Contact and MaxProp. In terms of social-based protocols there is no protocol already implemented by default.

Lastly the Visualization and Results consists in the obtaining the results data through reports that are generated using the report module. There are several types of reports that come by default, but it is possible to create personalized ones with the information that the user requires. Those reports can then be visualized through separate text files.

III. FRIENDSHIP PROTOCOL

A. The Concept

The Friendship protocol is a social-based routing protocol for DTNs which utilizes the ideas proposed in [9]. 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 personal and adjusted to take advantage of The ONE Simulator 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 Protocol

Time has a great importance 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 chronometer 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 timeslot in matter 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: \( \text{sim}_{-}\text{day} = \left\lfloor \frac{t}{86400} \right\rfloor \)
2: if \( t \) is a multiple of 86400 then
3: \( \text{sim}_\text{day}++ \)
4: end if
5: start_of_timeslot = (\( \text{sim}_\text{day} \)-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 \( \text{sim}_\text{day} \)
10: else
11: start_of_timeslot += 3 hours
12: end_of_timeslot += 3 hours
13: end if

Algorithm 2 – timeCorrect

Input: Time \( t \);
Output: Time \( t \_\text{corrected} \)

1: start_of_timeslot, sim_day = getTimeSlotInformation(t);
2: \( t \_\text{corrected}=(\text{sim}_\text{day}-1)*3 \) hours + (\( t \) - start_of_timeslot)
3: return \( t \_\text{corrected} \)

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 a node \( i \) to \( j \) is

\[
SPM_{i,j} = \frac{\int_{t_{\text{start}}(t)}^{t_{\text{end}}(t)} f(t) dt}{T},
\]

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 the period \( T \), then the SPM from a node \( i \) to \( j \) is

\[
SPM_{i,j} = \frac{\sum_{x=1}^{n} t_{\text{inter}x}}{2T},
\]

being \( t_{\text{inter}x} \) the \( x \)th intermeeting time value registered. 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 they followed the two-hop path \( \langle i,j,k \rangle \). 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 at 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 second stage, stage \( b \), starts when the first one ends and finishes when node \( j \) meets \( k \). Denoting the total number of sessions as \( n \), the definition of RSPM from a node \( i \) to \( k \) crossing \( j \) is

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

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

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

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, \( \text{sim}_\text{day} \) = getTimeSlotInformation(t)
2: \( t \_\text{corrected}=\text{timeCorrect}(t) \)
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
6: end_of_encounter(timeslot_index,otherHost) = 0 seconds and totalS(timeslot_index,otherHost) = 0 seconds
7: end if
8: intermeeting_time = start_of_encounter(timeslot_index,otherHost) - end_of_encounter(timeslot_index,otherHost)
9: previous_totalS = totalS(timeslot_index,otherHost)
10: totalS(timeslot_index,otherHost) = previous_totalS + (intermeeting_time)^2
11: SPM(timeslot_index,otherHost)= totalS(timeslot_index,otherHost) / (2*t_corrected)
12: endB(timeslot_index,otherHost,k)=t_corrected
13: if there are no records of encountering otherHost in timeslot timeslot_index then
14: initialize
15: startA(timeslot_index,otherHost,k)=0 seconds and endA(timeslot_index,otherHost,k)=0 seconds
16: end if
totalR(timeslot_index, otherHost, k) = 0 seconds²

16:    end if
17:    if endA(timeslot_index, k, otherHost) is superior
18:        than startA(timeslot_index, k, otherHost) then
19:        tb = endB(timeslot_index, k, otherHost) -
20:            startB(timeslot_index, k, otherHost)
21:        ta = endA(timeslot_index, k, otherHost) -
22:            startA(timeslot_index, k, otherHost)
23:        previous_totalR = totalR(timeslot_index, k, otherHost)
24:        totalR(timeslot_index, k, otherHost) = previous_totalR
25:            * 2 * ta + ta²
26:    end if
27: end for

Algorithm 4 – neighborLeft

Input: Time t; Node otherHost;
Output: (none);
1: timeslot_index, start_of_timeslot, end_of_timeslot, sim_day
2:     = getTimeSlotInformation(t);
3:     time_corrected = timeCorrect(t);
4:     if a timeslot overlap occurred then
5:         neighborLeft(otherHost, start_of_timeslot - 0.1)
6:     else
7:         end_of_encounter(timeslot_index, otherHost) =
8:     time_corrected
9:     SPM(timeslot_index, otherHost) =
10:    totalS(timeslot_index, otherHost) / (2 * time_corrected)
11:   for every other host k in the network besides the node that
12:     has just left
13:     endA(timeslot_index, otherHost, k) = time_corrected
14:     startB(timeslot_index, otherHost, k) = time_corrected
15: end for
16: 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 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
3:         threshold τ on the current timeslot then
4:         request RSPM values regarding itself towards every
5:             other node on which otherHost is the intermediate
6:         node
7:     true
8:     else
9:     false

Algorithm 6 is the last algorithm to be executed upon a new connection is 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
3:         send message m to otherHost and delete m from buffer
4:     true
5:     end if
6:     if the destination of the message is on otherHost friendship
7:         community then
8:         if otherHost’s friendship weight ω towards
9:             destination is superior to ours
10:        send message m to otherHost and delete m from buffer
11:        true
12:    end if
13:    false
14: return false
C. A dynamic threshold version

The protocol presented in the previous section was implemented respecting the idea that the threshold 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, there could be other ways to define each threshold.

An altered version of Friendship is also proposed in this work on which is thought that the threshold should be different for 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 instead be a portion of its best friend weight and not a fixed value that each node should adopt. 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 as the other stays as the Classic Friendship version.

IV. ANALYSIS OF RESULTS

The protocols chosen to be tested along with Friendship are the Epidemic, Prophet and BubbleRap protocols, which all have particular characteristics that might be drawbacks when compared with Friendship. In Phase 1, the primary objective is to tune BubbleRap’s parameters \( k \) and \( \text{Familiar Threshold} \) and Friendship’s \( \text{Friendship Threshold} \) 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 other 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 created messages 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.

There are three types of nodes circulating on top of the map, on a total of 114 nodes: pedestrians (90), trams (12) and buses (12). The main idea is that both pedestrians and vehicles possess an equipment like a smartphone with an integrated network interface that they use to exchange messages with others. Trams and buses have the movement pattern RMBM, 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 movement pattern WDM. 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 8 hours at work and after that they have a 75% probability to go socialize 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. The size chosen for each node’s buffer is 1 GB, which is a plausible value taking in account that a typical smartphone nowadays easily has around 8 GB of internal storage. 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 Mbps and limited to a communication range of 10 meters. The throughput is set to 10 Mbps as an average data rate which is constant for every position within the radio range\(^1\). In turn, the 10 meters range limitation may also seem inappropriate as there are currently access points that can reach up to 100 meters outdoors (e.g. a traditional 2.4 GHz access point), but, since the simulator does not take in account attenuations from obstacles like buildings, using a short fixed range results is a good approximation of reality.

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.

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 30.59%, an overhead ratio of 41.76 and an average delay of 14258.18 seconds, which

\(^{1}\) The ONE simulator handles radio transmissions by setting that the throughput is constant within the radio range and null for higher distances. This is actually a limitation of the simulator as in a real scenario there are generally significant attenuations along the signal path which reduces the throughput along the way.
approximately corresponds to 4 hours. In the case of the classic Friendship protocol, although it was hard to find a conclusion on 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 weight inverse, of 2 hours and 15 minutes, corresponding to the maximum delivery rate of 36.53%, an overhead ratio of 2.4 and an average delay of 18858.08 seconds (5.24 hours). For the dynamic version of Friendship, the dynamic threshold chosen as optimal was 80% of the best friendship, which corresponded to a delivery rate of 35.23%, an overhead ratio of 1.13 and an average latency of 18377.66 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 2. Both versions of the Friendship Protocol had positive performances in terms of the delivery rate metric, as no other protocol tested showed better results according to the confidence interval.

![Figure 2 – Delivery rate vs. message generation rate](image)

Regarding the overhead ratio metric, the results over the different message generation rate are showed in Figure 3. 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 3 – Overhead ratio vs. message generation rate](image)

The results regarding the average delay metric are presented in Figure 4. The general impression when analyzing the data is that the average message latency decreases for every protocol tested when the message generation rate increases, which is concordant with the idea that a significant part of the messages is discarded due to network congestion as several nodes have fully occupied buffers and have to discard 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 relative to each node.

![Figure 4 – Average Delay vs. message generation rate](image)

V. CONCLUSIONS

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.

Section 1 presented a brief overview of the DTN scientific field evolution, the motivation to the topic of this work and the scope of this paper. This was the starting point to all of the research and development done.
In Section 2, the fundamentals of DTNs are presented. This chapter starts with a brief description of DTNs and how routing schemes are evaluated, followed by a section solely dedicated to the typical network architecture of a DTN and after that, an analysis on the state of the art routing protocols is presented. In the end of this chapter, the existing simulators to test protocols for DTNs are referred and the one used in this document is analyzed in further detail.

In Section 3, the developed Friendship protocol is presented. In this chapter, it is possible to understand what is the main idea behind this protocol and how it was implemented, illustrated with the pseudo-codes of the most important algorithms and a flowchart. The Friendship protocol is based in two important concepts for social-aware routing protocols in DTNs: Friendship and Community. Node relations are always analysed 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 a second. 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 second version of this protocol on which the threshold is a portion of the current best friendship is proposed, meaning that friendship standards change from node to node.

In Section 4, the conditions on which the Friendship protocol was tested and evaluated are explained in detail. 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. Several conclusions were stated on this chapter.

As suggestions for future work for the topic, 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 the BubbleRap or the average number of friends per node for Friendship, as that would definitely allow a more precise tuning. Additionally, it would also be relevant to test the protocols in different scenarios to check how Friendship behaves under conditions different than of a MSN, the type of network which clearly it was designed for. Lastly, it could also be worthwhile to create a multiple-copy version of the Friendship Protocol, as that may result in significant performance gains.

REFERENCES


