# Writing on the Clean Slate: Implementing a Socially-Aware Protocol in Haggle

Mirco Musolesi
Dartmouth College, USA
musolesi@cs.dartmouth.edu

Pan Hui, Cecilia Mascolo and Jon Crowcroft
University of Cambridge, United Kingdom
[pan.hui|cecilia.mascolo|jon.crowcroft]@cl.cam.ac.uk

## Abstract

*Developing protocols and applications for opportunistic networking can represent a daunting task given the many aspects that must be taken into consideration, such as intermittent connectivity, smart choice among multiple interfaces and intelligent data storage. The implementation of these protocols can be based on generic layer-less communication frameworks that provide programming abstractions for the extraction and analysis of social, colocation and mobility information and allows data exchange by means of heterogeneous devices.*

*We propose Gently, a novel fully implemented solution which combines techniques of context awareness and social knowledge to concretely solve issues related to opportunistic forwarding. More precisely, Gently is born as the combination of the Context-aware Adaptive Routing (CAR) and the socially aware LABEL protocol. We discuss the implementation of our solution on top of the layer-less Haggle framework presenting the key design choices and the lessons learnt.*

## 1 Introduction

Recent wireless technologies have enabled novel communication paradigms capable of taking advantage of the presence of multiple interfaces on a single device and of the inherently opportunistic nature of the communication media. Furthermore, some of the ad hoc communication technologies, such as Bluetooth and Zigbee, have increased the ability of people to interact by following more natural social patterns.

A considerable number of research projects with the aim of designing protocols to support communication in these kind of social contexts have been carried out in the recent years. These research efforts are collectively referred to as opportunistic networking (also called *Pocket Switched Networking*) [13] and constitutes a particular class of delay tolerant networked systems [5]. Many of the routing protocols for this class of network are based on the exploitation of mobility [12] and colocation information, such as [9, 11], and social information, such as [8, 6, 2, 3]. In addition to routing protocol there is also a need for the provision of frameworks that are able to support programming abstractions for this kind of systems. A classic layer-based infrastructure might not be sufficiently flexible for opportunistic protocols that often interact with different components/layers of the system at the same time. Haggle [13] and the DTN2 [4] frameworks are recent examples of generic communication platforms for heterogeneous delay tolerant networks.

In this paper we present the implementation of Gently[1], a protocol which combines our experience with context-aware and socially-aware routing and practically shows how the protocol can be embedded into a DTN framework. More specifically, Gently is based on the Context-aware Adaptive Routing (CAR) protocol [11] and LABEL [8], a social-aware protocol. CAR is a natural fit for the Haggle vision, since it exploits prediction of people movement and colocation patterns to deliver messages in an asynchronous way. At the same time, LABEL is able to exploit social clustering in order to support message asynchronous delivery to members of a specific community. It represents a complementary mechanism to the prediction based technique offered by CAR. To summarise, the contributions of this paper are as follows:

- We present a new protocol (Gently) which combines a social-tag based method like LABEL and a predictive protocol like CAR to support opportunistic communication.
- We discuss the implementation of this solution according to a clean slate networking approach in the Haggle platform.

The remainder of this paper is organised as follows: in Section 2 we present the CAR and LABEL protocols. In

[1]The protocol is named after the Dirk Gently character in Douglas Adams' *Dirk Gently's Holistic Detective Agency* [1]. In the novel Dirk Gently uses a holistic approach, based on the (social) interconnectedness of all things; he follows (any) *car* that looks like it knows where it is going, because it will probably lead him where he wants to be.

978-1-4244-2100-8/08/$25.00 ©2008 IEEE

Section 3 we describe how the two protocols have been integrated in Gently. Section 4 contain the implementation details of the new protocol in Haggle and outlines the main design challenges. Section 5 focusses on the open issues in designing probabilistic protocols for communication in intermittently connected networks, while Section 6 concludes the paper.

## 2 CAR and LABEL

This section outlines the two protocols which are at the basis of our approach.

### 2.1 Context-aware Routing (CAR)

The design goal of the CAR protocol is to support opportunistic communication in intermittently connected mobile ad hoc networks. A detail description and the performance evaluation of this protocol can be found in [11]. The basic assumption of CAR is that a path between sender and receiver may not exist when the message is sent. Store-and-forwarding mechanisms are necessary to allow for the delivery of the message to the destination. The key routing problem is the selection of the carrier by means of an intelligent forwarding mechanism. The CAR algorithm is based on the application of forecasting techniques and utility theory for the evaluation of different aspects of the system that are relevant for making routing decisions.

CAR is able to deliver messages synchronously (i.e., without storing them in buffers of intermediate nodes) and asynchronously (i.e., by means of a store-and-forward mechanism). The delivery process depends on whether or not the recipient is present in the same connected region of the network (cloud) as the sender. If both are currently in the same connected portion of the network, the message is delivered using an underlying synchronous routing protocol to determine a forwarding path. If a message cannot be delivered synchronously, the best carriers for a message are those that have the highest chance of successful delivery, i.e., the highest *delivery probabilities*.

Delivery probabilities are synthesised locally from *context information*. More specifically we use the *change rate of connectivity* of a host (which, in our model, is used to measure the likelihood of it meeting other hosts) and its *colocation* with the message recipients.

### 2.2 Socially-Aware Forwarding (LABEL)

Identifying social information such as community membership about recipients can help to select suitable forwarders, and reduce the delivery cost compared to naive "oblivious" flooding.

The most simple socially-aware forwarding scheme is called "labelling strategy" [8]. The protocol is based on the assumption that each node has a label telling others about its affiliation/group, just like in conference badges. The strategy chosen is exclusively to forward messages to destinations, or to next-hop nodes belonging to the same group (same label) as the destinations. The evaluation of this protocol with different real data sets can be found in [8].

The problem with the simple "labelling strategy" is that the source needs to wait until the community members of the destination or the destination are at 1-hop distance from itself to start the forwarding. This may be not acceptable in many situations, since it may happen that the sender will never be in reach of a member of the community of the message recipient. Therefore, we propose an approach that combines CAR and LABEL. CAR can be used to reach a community, to route a message *inside* a community or when social information is not available. The details of our solutions are presented in the following section.

## 3 Gently

CAR was initially designed for unicast communication. We have relaxed this assumption in Gently and support anycast communication or, more in general, multicast towards *communities* by means of its store-and-forward routing mechanisms. Gently defines collective names, or, more precisely, names that identify groups of hosts: a recipient $i$ of a message can be a single host or a group/community of hosts (e.g., communities of individuals or people interested in messages of a certain type, like in content based routing). We adopt a simple solution: a generic name $i$ refers to a generic *class* of hosts which can be a singleton set, in the case of unicast, or a set with a higher cardinality, in the case of communities. The labels that are used to identify communities in LABEL are used in Gently as names for these classes of hosts. Every host may be assigned to a group identified by a certain label, if this information is available.

Let us assume that a host $H_z$ has to send a message to a generic host $H_x$ belonging to a community $C_y$. Let us also suppose that $H_z$ does not belong to $C_y$ in order to consider the more general case. The Gently forwarding process is based on the following algorithm:

- If $H_x$ is in reach, the message is delivered synchronously.
- If $H_x$ is not in reach, but a host belonging to $C_y$ is in reach (LABEL based routing), the message is forwarded to it; the host will store the message in its buffer. Gently will then use a CAR-like routing approach *inside* the community for delivering the message to the specified member of the group by means of the best carrier among the group members.
- If a host belonging to $C_Y$ is not in reach, the message is sent to the best carrier for $H_x$ that will store the message (CAR-like routing). The best carrier may be the node itself.

- If no information is available about the best carrier for the destination $H_x$, the message is sent to the best carrier for the community $C_y$. This host is chosen by evaluating the best known carrier for a host of the community $C_y$ that is characterised by the highest delivery probability.

This process is performed periodically and the number of retransmissions can be set by the developer, for example to limit the network traffic. If the recipient of a message is a group, the message is forwarded to any member of the group and then replicated in an epidemic manner among the members of the group. In scenarios where memory and energy consumption do not represent an issue or if it is necessary to increase the reliability of the delivery process, multiple copies can be used to reach a group and/or the message recipient. The investigation of this mechanism is outside the scope of the present work. In the next section we describe how Gently has been integrated into Haggle.

# 4 Implementation of Gently

## 4.1 The Haggle Platform

We now briefly present the aspects of the Haggle platform that are relevant to the implementation of Gently. Haggle is an example of a novel DTN non-layered software architecture. The architecture of Haggle is composed of six independent *Managers* forming a layer-less communication framework [13]:

**Data Manager** This entity is responsible for managing the data that are stored in a searchable repository of *Data Objects*, an abstraction used in Haggle to store data with a set of attributes.

**Name Manager** This entity is responsible for the mapping between different names identifying the user-level endpoints. The names of entities are stored as *Name Objects*.

**Forwarding Manager** This manager is responsible for delivering Data Objects to other hosts. It encapsulates a certain number of Forwarding Algorithms that are used to deliver the data to the final destinations. CAR is implemented as one of these algorithms. Applications can start a data transfer by specifying a set of Data Objects and a set of Name Objects. The Forwarding Manager then constructs a *Forwarding Object* that is a wrapper of a Data Object containing metadata needed to the forwarding operation. The metadata can also include expiration times and hop counts.

**Protocol Manager** This manager is only responsible for encapsulating and managing a set of *Protocols*. According to the Haggle terminology, a Protocol is defined as a method used to transfer a Forwarding Object between two nodes (i.e., to enable point-to-point communication).

**Connectivity Manager** This manager is responsible for encapsulating a certain number of so-called *Connectivities*, representing the available network interfaces. It provides support for neighbour discovery and for managing communication channels between two hosts. The Connectivity Manager provides the abstraction of *Neighbour* that is a potential next-hop by which a Protocol may know how to send data of certain types to specific Name Objects.

**Resource Manager** This manager schedules the various *Tasks* (i.e., operations) according to a predefined cost-benefit algorithm. Different algorithms can be plugged in by developers. All the outgoing and incoming network connections are proposed for the scheduling to the Resource Manager and executed according to their priorities based on the evaluation of their possible benefits.

All Haggle managers provide interfaces which are used for the communication and coordination among each others, including the Forwarding Algorithms, like Gently. In particular, our Gently implementation interacts with the Data Manager in order to store and retrieve Data Objects, with the Protocol Manager to obtain the list of the hosts that are currently reachable and with the Name Manager to manage the identities of the hosts and the users. The role of Forwarding Algorithms is to route the Forwarding Objects to the final destination(s). Gently is implemented as a pluggable Forwarding Algorithm for Haggle. For each Forwarding Object that has to be sent, the Forwarding Algorithm creates a *Forwarding Task* that is executed by the Resource Manager according to priorities evaluated with a decision algorithm that calculates the benefit of performing a certain task. In our implementation, we set the benefit to the maximum to guarantee the immediate scheduling of the task. When executed, the Forwarding Task causes the associated Protocol to send the Forwarding Object to the specified recipient. From a more practical point of view, Forwarding Algorithms must implement the `setForwardingTasks()` method that receives in input the list of the pending Forwarding Objects. These Forwarding Objects may have been created by the host or may have been received from other hosts and stored temporarily. Periodically, the Forwarding Manager invokes the `setForwardingTasks()` method. In the current implementation of Haggle, this interval is fixed, but this can be easily modified by developers.

## 4.2 Implementation Overview

Gently is implemented as a single Forwarding Algorithm for the Haggle platform. The implementation of this solution is presented in the following section. We designed a prototype implementation of Gently supporting one hop synchronous communication combined with the community based forwarding. From a practical point of view, in order to run Haggle with Gently, this needs to be added to the list of the routing protocols available for the forwarding process in the configuration file of Haggle. No modifications to the Haggle codebase is necessary. The main class of the

Gently package is `GentlyForwardingAlgorithm`; encapsulates the logic of the algorithm implementing the interface `ForwardingAlgorithm`. This class also implements the LABEL based mechanisms that can be activated/deactived by the developer. The constructor of this class launches three threads, responsible for forwarding the data and control messages for creating and updating the predictors.

When the `setForwardingTasks()` is invoked, the recipient of each Forwarding Object passed as a parameter is extracted. Then, the protocol verifies if the recipient is currently a neighbour (i.e., reachable at least through one of the available interfaces) and, if this the case, the object is forwarded to it. If a synchronous delivery is not possible, the forwarding process described in Section 3 is performed. If the message can be forwarded to a member of a certain community or to the Gently best carrier, the message is added to the list of Forwarding Objects that is managed by the Resource Manager. Otherwise, the message is forwarded to the best carrier currently in reach. We note that we had to provide support for *multi-regions routing*. In other words, a host may be reachable through different network interfaces. Different cost metrics can be assigned to different interfaces.

Haggle does not provide any mechanism for exchanging information periodically, like routing tables; however, this is essential for proactive protocols. We implemented the support for it by means of the classic threads synchronisation techniques available in Java and the available Haggle components and abstractions. In particular, the routing table messages are implemented as a special type of Forwarding Object. More precisely, we added a field in the Data Object that distinguishes these objects from those carrying data called `ProtocolCode`. Forwarding Object containing routing tables are characterised by a specific value of this variable. This field is used for searching among the Forwarding Objects stored in the Data Manager. Since the current version of Haggle does not support binary fields in the Data Object but only Java strings, we implemented two methods to serialise and deserialise the routing tables into/from strings (i.e., the `encodeRoutingTableIntoString` and the `encodeStringIntoRoutingTable` methods).

Each host maintains a routing table that is implemented as a `HashMap` containing entries that are instances of the class `GentlyRoutingTableEntry`. The entries of the routing table contains the `nextHopHostId` (that is used for synchronous routing), the `bestProbHostId` (for CAR-based routing) and the `community` fields (for LABEL-based routing). The routing table is indexed using `targetHostId` (i.e., the identifier of the destination).

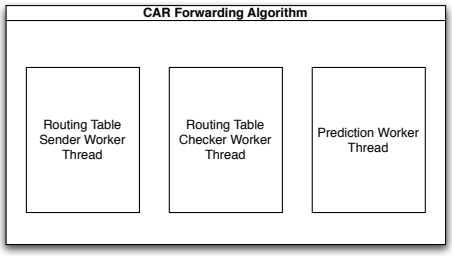


**Figure 1. Multithreaded structure of the implementation.**

### 4.3 Multi-threaded structure

Our goal was to encapsulate all the logic of the protocol in one class in order to simplify the integration of Gently in the platform. The multi-threaded structure of the class implementing the forwarding algorithm is presented in Figure 1. We now describe the role of each thread in detail:

**Prediction Worker** The `PredictionWorker` is responsible for updating the delivery probabilities by retrieving information about the context from other Haggle managers and by calculating the next step prediction using the Kalman filter based local predictors. More specifically, the `PredictionWorker` firstly updates the colocation predictor by checking if the host is a neighbour (i.e., it is reachable through at least one of the available interfaces). This is performed by requesting the list of Protocols to the Name Manager and then by checking all the reachable neighbours through a specific protocol. This logic is implemented by the method `isColocated()` that receives a Name Object as parameter. If a new node is discovered, a new Kalman filter predictor is initialised and added to the pool of the predictors[2]. The thread also updates the current change degree of connectivity and the related Kalman filter. Then, for each host, the combined weighted utility is calculated. The estimated delivery probabilities of all the nodes that are present in the routing tables are also updated. The other role of the thread is to check if the delivery probability of the host is higher than the one stored in the routing table and, if this is the case, to insert its identifier as best carrier and record its corresponding current delivery probability. Finally, this thread also checks if new names of the host have been added to the Name Manager, inserting these names to the routing tables for the delivery of the messages. A host can have different entries in the routing tables, one for each name. A different metric can be associated to different types of names (i.e., communication domains). An

[2]In the current implementation, no predefined size of the pool is set. However, a possible choice is to enforce a fixed size by removing the hosts of the pool with the lowest colocation values.

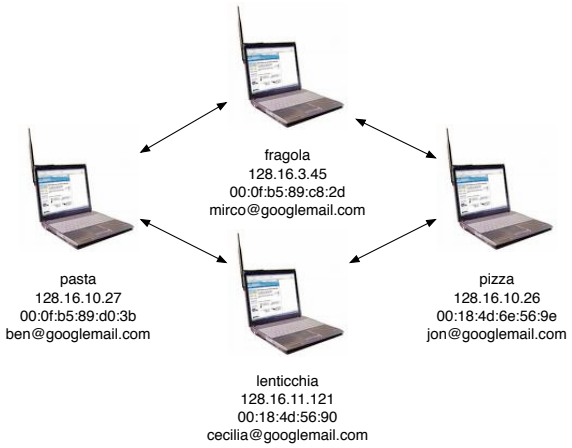


**Figure 2. Topology and addresses of the hosts of the Haggle testbed.**

alternative choice is to keep a name graph for all the names indicating the same host, but this does not allow for implementing specific calculation of the best path according to a given metric or to particular cost/benefit mechanisms.

**Routing Tables Sender Worker** The role of the `RoutingTablesSenderWorker` thread is the periodic transmission of the routing tables to the other nodes that are currently in reach. The routing tables are implemented as a special type of Forwarding Objects that are generated by the Forwarding Algorithm itself by constructing them as Data Objects. They are then sent as standard Forwarding Objects to all the neighbours on all the interfaces.

**Routing Tables Checker Worker** Another worker, implemented by the `RoutingTableCheckerWorker` thread, is responsible for checking if Forwarding Objects containing routing tables have been received. This is necessary, since an architecture based on listeners for particular types of messages (based on message filters) is not currently available in Haggle. This thread periodically checks if the Data Manager is currently storing routing tables by means of a filter on the `protocolCode` field. In theory, this should be done using a persistent filter that notifies an event handler, but this has not yet been implemented in Haggle. If Forwarding Objects containing routing tables have been received, these are deserialised and the information is used to update the local routing table. These threads are started as daemons (according to the Java terminology) when Haggle is launched. The interval of retransmissions of routing tables and updating of the predictors can be set by developers.

### 4.4 Functional Testing

In order to perform a functional test of the integration of Gently in Haggle, we have set up a testbed of four desktop computers equipped with 108 Mbps Netgear WG113T Wireless Adapters. The testbed is represented in Figure 2. We simulated disconnections at software level, testing different *virtual* topologies; one of these is shown in Figure 2, where even if the four hosts are physically in radio range, the host `pasta` is not able to communicate using synchronous routing to `pizza`.

We used a SMTP/POP application to test the platform. This application is composed of two components: a SMTP/POP proxy for the interaction with an email client (like Thunderbird) and SMTP and POP protocols (implementing the `Protocol` interface) inside Haggle that communicate with email servers. The SMTP proxy translates the email sent by the client into a Forwarding Object with one or more Data Objects, one containing the email itself and the others if attachments have to be sent. The POP proxy listens on port 110 for incoming connections from the user's client. When the client checks for email for the first time, the POP proxy inserts a new known name (the email address of the host) in the Name Manager. CAR is periodically checking if new names have been registered requesting the list of known names of the host to the Name Manager, as described above. The new names are then inserted in the routing table. In Figure 2 we indicate all the names associated to the hosts (DNS name, IP address, Ethernet 802.11 address and email address of the user). We ran different test to observe if the asynchronous delivery process was working correctly. We tested the routing tables dissemination, the store-and-forward and the selection of the best carrier mechanisms[3].

Since our Gently prototype relies on Haggle for the communication between pairs of nodes in ad hoc mode, the transmission delay between two hosts is not influenced by our implementation of Gently. Some measurements of these delays are presented in [13].

## 5 Lessons Learnt and Open Issues

In this section, we discuss some key aspects of our implementation experience in more details, outlining the lessons learnt, with the aim of providing a number of suggestions for the challenged networking community for the design of future opportunistic communication frameworks and protocols.

**Deterministic and Probabilistic Delivery Mechanisms** In Gently a host $h$ calculates (and keeps information about) its delivery probability $P_{h,i}$ for a certain number of hosts

[3]We also implemented a *carrier* testing mode. A host in carrier mode will have delivery probability set to 1 for all the hosts and then it will always be chosen as best carrier.

$i$ (or classes of hosts $i$). By exchanging the routing information, every host maintains information about a certain number of potential carriers for hosts/classes $i$. In case the delivery of the message is not based on a predicted or potential connectivity, but is rather *certain*, since the characteristics of the network infrastructure or, more in general, the future connectivity of the delay tolerant scenario is known and/or planned *a priori*, we refer to it as deterministic. In this case, the delivery probability will be equal to 1. For example, the delivery probability $P_{h,i}$ of a host $h$ connected to a host $i$ by means of a satellite link available during scheduled transmission slots will be equal to 1, since we know a priori that the delivery of the message will be possible in the future. Similarly, a bus that has a fixed and predefined route and that will deliver the message to a wireless Internet gateway will be associated to a delivery probability equal to 1 for the Internet-related addresses. With this model, we can treat and unify deterministic and probabilistic delivery mechanisms in our implementation of Gently, by exploiting the heterogeneous network interfaces provided by Haggle.

**Definition of Metrics and Costs** With respect to deterministic routing, the selection among multiple routes can be based on the the calculation of the shortest path considering different weights for links of distinct network domains. For example, a GPRS link should be characterised by a weight higher than the sum of the weights associated to several 802.11 links in order to avoid to be selected even if, from a topological point of view, it is only one single hop from the recipient. In any case, rule-based selection may also be enforced for prediction based and socially-aware routing decisions. We believe that the assignment of these costs, for example using market-based strategies, represents an open issue for the opportunistic network community.

**Automatic Selection of the Protocols** Another interesting problem is the definition of the automatic selection of protocols according to the context characteristics and the application scenario requirements. The routing decisions may be rule-based (i.e., static) or can rely on the evaluation of the current state of the system (i.e, dynamic) in terms of current resource availability, connectivity, security and cost. In fact, users may want to use the cheapest transmission medium (for example, a mobile phone carrier instead of a GPRS link). This information can also be included in the bundles and evaluated when a routing decision has to be made. For example, hosts may not be selected even if they have the highest delivery probability for economic or security reasons according to the sender requirements. Haggle provides a natural support for the implementation of this mechanism by means of the Resource Manager that evaluates the associated cost and benefit of taking a certain forwarding action. However, we also believe that a more expressive set of primitives are necessary to implement more complex policies. A possible deployment scenario is the support of emergencies, for example in case of terrorist attack: in these situations, the framework should be able to select automatically epidemic routing algorithms (the standard one proposed by Vahdat and Becker [14] or more optimised ones such as [7]) instead of probabilistic algorithms without any trade-offs in terms of transmission costs, security, etc. We are currently investigating these aspects, also considering a possible declarative routing approach [10] to define the rules of the selection of the forwarding strategies.

# 6 Summary

In this paper we have presented the Gently, a routing protocol for Pocket Switched Networks based on the combination of the CAR and LABEL, implemented on top of the Haggle framework. We have discussed the lessons learnt from the development and integration of the routing components, outlining the open issues and research challenges.

**Acknowledgements**

The authors would like to acknowledge James Scott for his comments and acknowledge the support of EPSRC through project CREAM.

# References

[1] D. Adams. *Dirk Gently's Holistic Detective Agency*. Pan Macmillan, 1998.

[2] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco. Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks. *Journal on Selected Areas in Communications*, 26(5), June 2008.

[3] E. M. Daly and M. Haahr. Social network analysis for routing in disconnected delay-tolerant MANETs. In *Proceedings of MobiHoc'07*, pages 32–40, New York, NY, USA, 2007. ACM.

[4] DTN-2 Reference Implementation Version 2.5, October 2007. http://www.dtnrg.org/wiki/Code.

[5] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of SIGCOMM'03*, August 2003.

[6] J. Ghosh, S. J. Philip, and C. Qiao. Sociological Orbit aware Location Approximation and Routing (SOLAR) in MANET. *Elsevier Ad Hoc Networks Journal*, 5(2):189–209, March 2007.

[7] Z. J. Haas and T. Small. A new networking model for biological applications of ad hoc sensor networks. *IEEE/ACM Transactions on Networking*, 14(1):27–40, 2006.

[8] P. Hui and J. Crowcroft. How small lables create big improvements. In *Proc. IEEE ICMAN*, March 2007.

[9] A. Lindgren, A. Doria, and O. Schelén. Probabilistic routing in intermittently connected networks. In *Proceedings of SAPIR 2004*, August 2004.

[10] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: Extensible routing with declarative queries. In *Proceedings of ACM SIGCOMM'05*, August 2005.

[11] M. Musolesi, S. Hailes, and C. Mascolo. Adaptive Routing for Intermittently Connected Mobile Ad Hoc Networks. In *Proceedings of WoWMoM'05*. IEEE press, June 2005.

[12] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of WDTN'05*, pages 252–259, New York, NY, USA, 2005. ACM.

[13] J. Su, J. Scott, P. Hui, J. Crowcroft, E. de Lara, C. Diot, A. Goel, M. Lim, and E. Upton. Haggle: Seamless networking for mobile applications. In *Proceedings of Ubicomp'07*, pages 391–408, September 2007.

[14] A. Vahdat and D. Becker. Epidemic Routing for Partially Connected Ad Hoc Networks. Technical Report CS-2000-06, Department of Computer Science, Duke University, 2000.