



Project IST 027918

An Innovative Paradigm for Autonomic Opportunistic Communication

Haggle

Deliverable 3.1: Interim release of Haggle. Analysis report of the trial and specification of the next trial

Instrument:	FET Integrated Project
Strategic objective addressed:	IST-2004-2.3.4 - Situated and Autonomic Communications
Due date of report:	31 st December 2006
Actual submission date:	20 th February 2007
Start date of project:	January 1 st 2006
Duration:	48 months
Project Manager:	Christophe Diot
Authors:	Fredrik Bjurefors, Oskar Wibling, Christian Rohner, Kaustubh Phanse, Christophe Diot, Silvia Giordano, Augustin Chaintreau, Salvatore Vanini, Eiko Yoneki, Jon Crowcroft,
Revision:	v1.0

Project co-funded by the European Commission within the 6th Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	✓
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	


	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

Table of Contents

1 INTRODUCTION 3

2 INITIAL TRIALS..... 4

2.1 iMotes experiments..... 4

2.2 Experiments in the Xen-Hagggle testbed 4

 2.2.1 Preliminary Trial 5

 2.2.2 Hagggle traffic generator 5

 2.2.3 Structure of the benchmarking suite..... 5

2.3 Experiments with Applications 6

2.4 Implementation in .NET 6

3 LESSON LEARNED 8

3.1 Limitations of iMote experiments 8

 3.1.1 Code overview 8

 3.1.2 Potential implications 9

 3.1.3 Discussion 10

 3.1.4 Conclusion 10

3.2. Java implementation in Unix..... 10

 3.2.1 Information Ageing and Garbage Collection 10

 3.2.2 Dynamic Cost/Benefit Analysis 11

 3.2.3 Forwarding Algorithms 11

 3.2.4 Attribute dissemination..... 11

 3.2.5 Native support 11

 3.2.6 Security 12

3.3 Java implementation in Windows..... 12

4 SPECIFICATION OF THE SECOND TRIAL 14

4.1 Pre-requisite 14

 4.1.1 Usability 14

 4.1.2 Resource manager..... 14

 4.1.3 Connectivity manager..... 15

 4.1.4 Forwarding algorithms..... 16

 4.1.5 Web application 16

 4.1.6 Windows mobile specifics..... 16


4.2 Second Trial..... 16

5 CONCLUSION 18

Appendix A: The iMote experimental environment

Appendix B: Hagggle clean-slate networking architecture

Appendix C: Implementation in .NET

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

1 Introduction

The Hagggle project is built on a close interaction of long term research and experimental evaluation. Various aspects in Hagggle require experimental evaluation: transfer opportunities, performance of communication technologies, usability, etc. We started the experimental evaluation with connectivity measurements using iMotes, but also implemented the INFANT Hagggle architecture both in Java and .Net.

The first experimental need was to evaluate the feasibility of opportunistic ad-hoc transfer, both in term of frequency of opportunities, and duration of contacts. Given battery problems with WiFi, we opted for iMote based experiments (described in the next section). iMote provided us with a lightweight, small form factor, way to measure mobility pattern in order to achieve our research objectives for the first two years in Hagggle. However, iMotes are not sophisticated enough to run Hagggle applications.


Note that iMote data sets have been extensively published and have been made available to the research community through the CRAWDAD repository.

Therefore, the strategy for the first year has been to perform multiple iMote experiments and to develop in parallel two INFANT Hagggle implementations: one in Java, and one in the .NET framework. We run the implementations on Linux, Windows XP, and Windows Mobile. As a consequence, the small scale experiments allowed us to validate our implementations. We did not perform an extensive performance evaluation. Lessons learned are not in term of performance, but in term of what to do next and how to design the next implementation.

There are multiple reasons for not doing performance measurements on INFANT Hagggle yet:

- We don't understand clearly the characteristics of human mobility and more experiments are needed to understand the basics of human mobility, and to design appropriate forwarding algorithms and scenarios.
- We could not find hardware that was stable enough, and which batteries would last long enough, to perform such an experiment. We tried Dell AXIM, Nokia 770, etc. But all devices we tried could not provide trustable and reliable performance data.

This document is structured as follows: Chapter 2 sheds light on the iMote experimental environment, as well as the initial INFANT Hagggle implementations. In Chapter 3, we discuss the lesson learned, both with iMote and with our implementation. These lessons will help us design CHILD Hagggle that will be more stable and reliable than INFANT Hagggle. In the first part of Section 4, we identify what has to be developed first for the CHILD release. In the second part of Section 4, we define the second trial, and justify changes to the current workplan in order to maximize the chance of migration from iMote experiment to Personal device-based experiments.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

2 Initial trials

2.1 iMotes experiments

Pocket Switched Networks (PSN) make use of both human mobility and local/global connectivity in order to transfer data between mobile users' devices. This falls under the Delay Tolerant Networking (DTN) space, focusing on the use of opportunistic networking. One key problem in PSN is in designing forwarding algorithms which cope with human mobility patterns. We decided to take an experimental approach to this problem. We present an experiment measuring forty-one humans' mobility at the Infocom 2005 conference. The results of this experiment are similar to our previous experiments in corporate and academic working environments, in exhibiting a power-law distribution for the time between node contacts.

We have chosen this article as it describes best our approach to the measurement of human mobility. Infocom 05 was the initial large scale experiment. Since the official start of Hagggle project (January 1st, 2006), we have performed multiple experiments in various environments. The most noticeable ones being Infocom 06 and CoNext 06 with 100 iMotes and social information about participants. This deliverable being about trials, but not about the analysis of the data, observation made from the data collected in all experiments will be described in a companion deliverable, and in particular in D5.1.

2.2 Experiments in the Xen-Hagggle testbed


We have created a testbed to emulate a mobile opportunistic network and conduct repeatable tests in a controlled and easy to manage environment. The Xen virtual machine monitor is at the core of the testbed. Xen supports execution of multiple guest operating systems (or emulated Hagggle devices) that are monitored by a Xen host system.

Currently, the testbed uses Xen 3.0.3 and a host system that runs on Fedora Core 5 on a Dell Dimension 5100 with a Core Duo processor with 4 GB RAM. The host system requires 256 MB of RAM and 10 GB hard disk space plus 2 GB per guest system. The host system consists of scripts to manage the guest systems, Uppsala University's APE tools, GNU Java, and simple database. In the future, we plan to test SQLite for database and compare the memory consumption. The goal is to eventually have a test set-up that is consistent with the system that will be used on real Hagggle handheld devices.

Each guest system uses 32 MB of RAM and the Xen monitor uses 32 MB. A non-PAE-enabled (Physical Address Enhancement) kernel, like the FC5 kernel, only supports 3228 MB of RAM, which is not enough to implement larger networks (100 plus nodes). In the future, we will use FC6 and a Copy-on-Write (CoW) implementation to manage a large number of guest systems by minimizing creation time, maintenance, and hard disk space requirements.

Ad hoc Protocol Evaluation (APE)

APE is a Linux distribution created with the goal of making the process of performing complex real-world tests as easy as possible. It focuses on smooth deployment, high ability of customization and ability to easily run several protocol implementations for comparisons. The Xen-Hagggle testbed uses the main APE script to interpret the scenario

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

file (connectivity) and to collect logs. Many changes have been made to make APE work better with the Xen-Hagggle configuration.

The dynamic connectivity or topology changes in a network are emulated by filtering traffic with IP tables according to the APE scenario file. We have implemented a `Trace2scenario` script that converts a real world connectivity trace (e.g., Hagggle iMote trace) into an APE scenario file; it also adds start and stop procedures and traffic generation during the emulation.

2.2.1 Preliminary Trial

We started by using a simple network of three emulated Hagggle nodes with random connectivity (individual links are connected about 50% of the time). We have implemented a simple messaging application where a source node generates PING messages and the destination responds with a pong message. Epidemic forwarding was used to forward messages. Source and destination nodes were chosen at random. The average one-way message delivery ratio was about 75%. However, the round-trip message delivery ratio (i.e., both PING and corresponding PONG message delivered successfully) was 2.5%.

We are currently testing a slightly larger network with 10 Hagggle nodes. To properly isolate and address potential problems in both the Hagggle architecture and the implementation of INFANT Hagggle, we propose to first use a set of well-structured tests. The selection of the first tests will comprise a benchmarking suite. Acceptable levels for delivery ratio and end-to-end delays will be set in order to determine if a version of Hagggle has passed a certain test or not. We require that Hagggle passes each test in several repeated experiments.

When we reach a level where INFANT Hagggle works satisfactorily in terms of delivery ratio and end-to-end delay, we will move on to tests generated from real-world mobility traces.

2.2.2 Hagggle traffic generator


We have written a Java application called *HagggleMessageGen* for the purpose of generating Hagggle traffic for the tests. *HagggleMessageGen* communicates, via TCP, with a Hagggle application and can thereby insert forwarding objects destined for a certain node. The forwarding objects we insert contain a PING message and when the final destination receives such a message they generate a reply, a PONG. In addition to delivery ratio and end-to-end delay, we can also measure the two-way latency or the round-trip time.

The traffic generator takes as input parameters the number of nodes to PING, initial delay, run time, and PING interval. The nodes to PING are then selected randomly from the set of available nodes. The selected nodes are used throughout the scenario.

2.2.3 Structure of the benchmarking suite

For the set of benchmarking tests we propose the following.

- Use the following basic topology types: line, grid, ring, star (see Figure 1).
- For each of the topologies mentioned above, vary the number of nodes involved, starting with 3-4 nodes and then increasing.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

- For each topology type, and number of nodes, vary the connectivity pattern as follows: We start by running experiments with full connectivity, then we move on to varying connectivity in a controlled way and finally we use random connectivity.

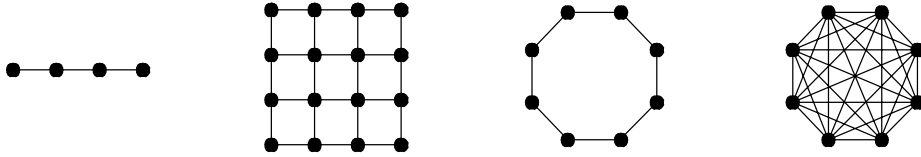


Figure 1: Basic topology types

By varying connectivity in a controlled way we mean that we select the sequence of link changes manually in order to test a certain behavior. As an example, consider the following circle topology: A-B-C-A. We may then use a scenario in which node A wants to send packets to node B. During the course of the scenario, the link between nodes A and B goes down during some time interval. The aim of the test is to see how the forwarding of packets from A to B is affected when the link break occurs.

Varying the connectivity randomly will be done using different distributions as well as using the Hagggle iMote connectivity traces.

For all combinations of topology, number of nodes, and connectivity patterns, we will construct scripts to facilitate the generation of APE scenario files. These APE scenario files are the ones used during test execution to start traffic generation and emulate connectivity changes in the network. The traffic will be generated using the Hagggle traffic generator and will be varied in terms of nodes to ping as well as the message intensity.


The testing procedure described above will allow us to examine and evaluate how Hagggle handles real-world scenarios in contrast to the benchmarking tests. Then, we will address issues in the implementation and additionally use the results to identify parts of the architecture that may need to be modified or extended for Hagggle to cope with real-life situations.

2.3 Experiments with Applications

The document in Appendix B describes several experiments using applications like email and web browsing. The experiments are performed with the Java implementation on two laptop computers running Windows XP. We refer to Appendix B for details.


2.4 Implementation in .NET

The document in Appendix A describes an implementation of Hagggle in .NET platform over Microsoft Windows Mobile 5. The prototype has been developed following the main architecture guidelines. All six managers are enhanced to adjust to the .NET platform and resource-constrained devices. Connectivity Manager is extended with Bluetooth and SMS conductivities, which support the email applications. We report the implementation details

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	 The logo for 'Hagggle' features the word in a white, sans-serif font. The letters are slightly shadowed and appear to be resting on a surface of golden sand dunes under a clear blue sky.
--	--	--

and problems run into. Such issues include six threads for managers running simultaneously over resource constrained devices, no free libraries in .NET to manage Bluetooth, WiFi and GPRS properly, and no support of multi-homing in Windows Mobile 5 because of security features.

Furthermore, the performance evaluation of the prototype is reported, including Bluetooth connectivity, forwarding algorithms and performance of Data Manager. Various experiment results include difficulty of scalability with Bluetooth connectivity, and lack of garbage collection of data objects and more efficient searching algorithms.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

3 Lesson learned

3.1 Limitations of iMote experiments


This section discusses our experiences with conducting an iMote mobility experiment at CoNEXT 2006 in Lisbon, Portugal. The goal of this experiment was to conduct a large scale experiment with 80 mobile nodes and 14 stationary long range devices in a conference environment. This was our second attempt of this type of experiment due to a software bug causing the motes to reset making the previous results unreliable. Our objective was to try to understand what was causing the problems with the previous version of the code in order to reuse as much as we could in order to minimize adding new bugs. Unfortunately, we experienced many unusual problems related to the TinyOS platform due to the lack of documentation that ultimately forced a total rewrite of the code to facilitate a reset free experiment. However, this rewrite also exposed another problem with how the motes had been assumed to operate in sleep mode that were, due to lack of time in the preparations, not revealed before the start of the CoNEXT experiment.

3.1.1 Code overview

We first looked at the use of the timers. Instead of firing a timer every second just to check if the inquiry interval had past, we allowed it to be idle for the inquiry interval (120 seconds). The procedure for scanning followed the basic idea of the previous version of the code. The iMote started by first scanning all available bluetooth devices and adding the responses to a memory structure (*contactList*). If a responding iMote was already in the *contactList*, its values were updated (e.g., end time). Otherwise, it was the first sighting of this contact and the appropriate values were added. After adding contacts to the *contactList*, the ones that were in the list but did not respond to the current inquiry were handled. We had what was called a "skip slot" to allow iMotes that had not been seen during a scan to remain in memory for one interval. If the iMote appeared in the next scan, the iMote's skip slot was reset and the iMote remained in the *contactList*. If the iMote was not seen after two consecutive intervals, it was assumed to be lost and therefore a log entry was written to permanent flash storage.

To protect against data loss in case of hardware resets (battery coming loose, bad wiring, etc.) we changed how data was written to permanent flash storage. Previously, the main idea was that data would be written to flash once a contact went away. However, in the unlikely event of a contact lasting for the duration of the experiment, the contact would never be recorded in flash. Also, in case of a reset, all knowledge of current contacts would be lost. The old code tried to protect against this by writing a temporary snapshot of the *contactList* to flash every 600 seconds.

In the CoNEXT code, the writing of data to flash was changed so that immediately when a new contact is made, a start time of that contact is written to flash. This eliminates the need for the temporary snapshot. The offset on the flash where the contact was stored is kept in the contact list in memory so that the entry can be updated when the contact goes away. We used the FlashFS implementation that was used in previous experiments but experienced unexpected behavior. Whenever we used the "seek" file system call to update an entry, the pointer that was returned was corrupted and contact entries were overwritten with incorrect data. To resolve this we implemented our own way of writing to flash by

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

directly accessing the blocks where the data was stored. By keeping track of the file pointer data in this way we were able to directly update the entry as opposed to reading the flash and searching for the contact every time we wanted to make an update.

3.1.2 Potential implications

In this section we briefly highlight some of the bugs we caught and the potential issues that may have an impact on the results from previous experiments and new experiments.


The FlashFS implementation had some bugs which caused the iMotes to reset at random times when there were many sequential writes to flash. This was resolved by implementing direct writes to the flash at the block level. This method bypassed all the unnecessary steps that were required in the implementation of FlashFS.

The random function that TinyOS provides was not seeded (i.e., always using the seed '1') thereby causing all iMotes to have exactly the same "random" offset. This was confirmed by looking at previous experiment logs and observing all the scan time intervals are exactly the same. This was fixed by seeding all the iMotes with their MAC addresses.

There were multiple buffer overflows associated with the contactList which contributed to the number of resets we saw in previous experiments. We took great care to make sure that these overruns were removed and believe that there are no bugs in the latest version of the code.

In the implementation of the bluetooth interface code (BTLowerLayersM.nc) the variable *MAX_INQUIRY_REQUESTS* was hard coded to eight. Having this in the code basically limited the total number of inquiries received and is what probably led to the assumption that a five second scanning interval was sufficient to capture all available bluetooth devices in the vicinity. We observed that this short scanning interval had issues with seeing the same devices for a longer period of time when there were lots of nearby iMotes. Whichever device was able to respond to the iMote while the inquiry was in progress was logged, otherwise the contact was removed from the contactList.

While investigating the logs from the CoNEXT experiment, we noticed that there were very short contact times between internal contacts (i.e. other iMotes) as opposed to external contacts (i.e. external devices like phones and PDAs). This discrepancy was not seen in previous experiments. Our first instinct was that these external devices had more power and capability to respond to all inquiries that it received. However, after further investigation, we attributed this problem to the sleep mode of the iMote which can be identified by observing the number of seconds in between flashes of the blue LED. If the iMote is awake, the blue LED will flash around every 3 seconds and will always respond to inquires. If the iMote is in deep sleep mode, the blue LED will flash every 15 seconds and will only sometimes respond to inquiries. To understand what was causing this difference in the sleep mode behavior we did comparisons between the new and old code. We found that only the new code used the sleep mode claimed to be used in the older code. Previously this sleep mode was assumed to not affect a mote's ability to respond to inquiries. If the sleep mode is disabled in the CoNEXT code, by calling *WakeUp()* and removing *GoToSleep()*, the issue with short contact times is resolved. For some reason sleep mode must have been unintentionally disabled in the previous code, leading to the wrong assumption that the sleeping did not affect the ability to respond to inquiries. Unfortunately, we were not able to find this out before the CoNEXT experiment.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

3.1.3 Discussion

From our experience and findings it is clear that the sleep mode of the iMotes can not be used when the iMotes must be able to respond to inquiries. It does seem that the iMote battery lifetime was longer for CoNEXT compared to previous experiments, but sleep mode operation is only suitable in cases where the motes should only reliably inquire non-iMote devices.

Furthermore, we need to have a better understanding of what will happen to our results if the scanning interval is increased. We performed a test in the conference room during one of the sessions and set the scanning interval to different values. We found that having an interval of thirty seconds allowed enough time for all nodes in the area to be able to respond to the inquiring iMote. However, from the previous findings of sleep time with regards to response, it may be that the iMotes were in a deep sleep and were not able to respond quickly enough to the 5 second inquiry. Care should be taken when modifying this value. If we increase this value, it will lead to more nodes being logged in our contactList, but also it may cause more contacts to be missed because they will be scanning at the same time. Increasing this value will also cause a considerable reduction in the duration of the experiments due to the load inquiries place on the battery. An experiment using two iMotes where one is inquiring and one is listening could answer some of these questions.

3.1.4 Conclusion

This document summarizes the issues that we experienced while conducting the iMote experiments for the Hagggle project. The programming of the iMotes proved to be very time consuming and frustrating due to the fact that there is virtually no documentation on these devices. We have removed all of the bugs that cause the iMotes to randomly reset and also identified why iMotes were not responding to inquiries.


3.2. Java implementation in Unix

The current INFANT Hagggle architecture is by intention kept simple and it is the plan to learn from the limitations of this architecture to identify improvements for the next (CHILD) Hagggle architecture.

In this section we go through some of the issues found while working with the Java implementation in a Unix environment. The intention is not to specify a new iteration of the architecture, that is a separate task in WP1, but to provide directions for that task and to specify trials that can measure the proposed improvements.

3.2.1 Information Ageing and Garbage Collection

As Hagggle opportunistically forwards on different connections and with different protocols, the Task Manager issues several Forwarding Objects for the same Data Object to be forwarded, one for each known name of a destination. Depending on the availability of connectivity, protocols, and the chosen forwarding strategy, some of the Forwarding Objects are scheduled as Forwarding Task. Once a Forwarding Task is executed, it is removed, but not the non-executed ones, and not the Forwarding Object itself as long as there are other Forwarding Objects referring to the same Data Object.

	Deliverable D3.1: Interim release of Haggle. Analysis report of the trial and specification of the next trial	
--	---	---

This leads to a continuously increasing number of Forwarding Objects and Forwarding Tasks in the system that occupies resources.

We therefore see a need for *ageing mechanisms* and *garbage collection*.

3.2.2 Dynamic Cost/Benefit Analysis

All outgoing or incoming network operations in Haggle are proposed to the Resource Manager and executed only if/when the Resource Manager chooses; they are not necessarily executed in order or at all.

A Task comprises a method of accomplishing the work, the benefits of achieving the Task, and the costs of performing the Task. This definition is deliberately abstract so that the Resource Manager can compare between different possible actions while knowing little about the actual mechanisms or details of formulating or carrying out Tasks. Both the costs and benefits of Tasks are re-evaluated by the Resource Manager each time a Task is considered for execution, using callback functions provided by at Task creation time. In order to optimize cost/benefit analysis of each Task and thus save time, its evaluation should be performed only if parameters' values changed.

Most of the cost, benefit, and thresholds are static in the current implementation of Haggle. One of the few task using dynamic benefit values, the neighborhood discovery task, shows that the concept works and hints at the possibilities of dynamic values and we therefore propose to consider dynamic cost, benefit, and threshold values.

3.2.3 Forwarding Algorithms

Forwarding in INFANT Haggle is intentionally kept simple. A Forwarding Object is only forwarded to a nearby node if one of that node's Name Object is listed as a destination in the Forwarding Object itself. In other words, information distribution is dependent on node mobility.

An epidemic forwarding scheme sending Forwarding Objects to every neighbor is implemented for testing. However, epidemic forwarding is not efficient, and routing loops can be expected in larger scenarios.

We therefore see a need for more sophisticated forwarding algorithms. We have proposals for self-limiting epidemic forwarding, forwarding with network coding, and probabilistic context based forwarding.


3.2.4 Attribute dissemination

Forwarding of data based on context information, as opposed to a specific destination, requires the dissemination of the attributes that a potential recipient might be interested in.

Context based forwarding requires decision based on attributes. The exchange of attributes needs some careful consideration. Name objects are not sensible because of the merging and creation of many Forwarding Objects. Given the community structure observed in many of the real-life traces, the use of communities as a context is of particular interest.

3.2.5 Native support

Haggle aims to support and embrace the use of many different networking technologies at the same time. The job of the Connectivity Manager in Haggle is to encapsulate a number

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

of Connectivity Objects. Each Connectivity must support a well-defined interface including functionality for neighbor discovery, opening/using/closing communication channels, and estimating the costs of performing network operations. The Connectivity must interface with the underlying hardware to provide this functionality.

There will be one Connectivity instance per instance of a network interface on a node. For now, native support is only provided for a limited set of network interfaces on specific operating systems. By moving towards smaller platforms like PDAs or mobile phones, new native drivers to support Connectivity interfaces are needed.

3.2.6 Security

The current Hagggle architecture does not include any security mechanisms. Basic signature and encryption mechanisms would be beneficial for many applications. An analysis of threats to new forwarding strategies such as network coding or probabilistic forwarding should be started.

3.3 Java implementation in Windows


Hagggle realizes the concept of opportunistic networking as data exchange take place during connection “opportunities”, that is when two or more mobile devices get in wireless range due to the mobility of their users.

Even if the approach of Hagggle is orthogonal to the one used in the Internet, we could have some scenario in which the opportunistic approach make reasonable to use an available Internet connection. For this reason we implemented the management of the Internet connection that acts as a bridge for the Hagggle applications. On the other side, the Hagggle network could be the only way to reach the Internet for an Internet application, and we did not prevent this use.

During the implementation process of Hagggle another consideration raised: although the GPRS connectivity is not related to the opportunistic vision, it could be a valid backup solution if no connectivity is present. Think for example to a client application for sending/receiving email: GPRS can be used for checking new email existence but the downloading of a large email can be executed when we are in the range of an AP. For this reason we decided to extend the topologies of network connections handled by Hagggle and decided to provide a mechanism for the management of the GPRS connection.

An important lesson learned from the implementation concerns the computational overhead imposed by Hagggle that is particularly evident on devices with limited computational resources. Currently, Hagggle requires significant system resources to execute its tasks. The use of a Resource Manager that continuously iterates between tasks and performs a cost/benefit analysis imposes significant overhead on the device it runs on. The use of DO filters to search for data implies some overhead over methods such as listening on a socket. For both of these cases we think that a sort of optimisation work should be done and tests on small devices (such as PDAs or smart phones) should be conducted to check if the Resource Manager could effectively support the execution of this large number of tasks.

Working with the implementation also suggested some considerations about performance.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

Currently the management of the connectivity is not optimised and this results in a scarce level of performance perceived by the user. Significant overhead is carried during the switch between infrastructure and ad hoc modes, particularly when switching to infrastructure mode if the DHCP handshake incurs packet loss. DHCP mechanism is one of the bottlenecks in the management of the connectivity and methods different from the standard should be investigated to decrease time lost during the switching process.

Another consideration that emerged during the implementation is that there are technical issues directly related to the operating system implementation that affects some architectural decisions.

Regarding to the Windows implementation, one of these issues is connected to Media Sense, a mechanism that checks the physical connection and invalidates the TCP stack if the connection is not present. Since Media Sense can remove the static IP address used in adHoc mode without warning, it affects the accept() call on a socket previously instantiated as it cannot detect this. The consequence is that we had to adopt a solution to invalidate the socket when the IP turns bad.

Another example is the suspension of the Windows Wireless Zero Configuration Service. This service disrupts Hagggle's attempt to use the wireless card in adHoc and infrastructure modes and has to be stopped before start-up.


A significant example of bad interaction with some Windows services is that Hagggle does not perform well with firewall services as it quickly toggles the wireless interface. This determines the disabling and stopping of any firewall services before start-up.

Finally, we experimented some difficulties with some WiFi adaptors mainly related to the switch between infrastructure and ad hoc modes (like problems in getting the MAC address of the wireless card or difficulties in setting the proper operating mode) that suggested us to conduct comprehensive tests to verify the compatibility with any wireless interface.

An important lesson we learned from the implementation is that usability is a key issue if we want to disseminate the new communication paradigm of Hagggle (that is the autonomic and opportunistic communication to mobile end points, which are intermittently in reach of other devices or networks). In this sense a lot of work should have to be done to improve the user experience, i.e. by proactively notifying the status of a message to the user or by advertising the presence of a neighbour.

The last consideration regards security. Although in the current implementation security and privacy have not been addressed as key concerns, there are particular security and privacy issues that cannot be ignored. One of these is the breach of privacy determined by the exposure of the full name graph (that can contain sensitive information) of a node to everyone who can see an FO with the graph.

Another issue has to do with neighbour discovery protocols, since the nature of the information beacons by devices is their identity. This could allow tracking of the user.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

4 Specification of the second trial

4.1 Pre-requisite

During the process of implementing Hagggle, we identified a number of research issues that should have to be investigated in the future releases, both in terms of new features needed and in terms of optimizing the algorithms used. We describe them in the next sections.

4.1.1 Usability

Currently the process of starting Hagggle is not very simple and immediate. The user has to manually start/stop services that handle the connectivity, set up parameters in the configuration file, and install some libraries in appropriate locations. This results in a bad usability that could discourage the usage of Hagggle. For this reason it could be nice to have an installer that automatically sets up the libraries and files, detects the list of connectivity of the device and update the configuration file consequently. Then, once Hagggle is started, it automatically disables/enables the services needed to run it properly.

Usability is a key issue for letting user get used to the new communication paradigm of Hagggle. In this sense a lot of work should have to be done to improve the user experience. At the moment, the email application shows that the email is sent as soon as the Hagggle proxy receives it. However, this is of course untrue. While Hagggle knows more information about the status of the message, this is not presented to applications. In the future a proactive notification will surely augment the user experience.

Furthermore, the process of deciding which tasks to execute, involves considerations on monetary costs, energy consumption, time spent on a network and number of bytes exchanged, and estimation of the importance of the task to the application and to the user. Providing a configuration interface to assist the user in manually set up these preferences will be a further step in usability. Furthermore, assisting the user in trouble shooting (i.e. what preferences should be changed) will improve the key concern of usability.


4.1.2 Resource manager

The current implementation of the Resource Manager performs the procedure of selection and execution of the best task using a fixed time interval (every 100 msec). Tests should be conducted in order to calibrate this period. Moreover, this period should be configurable to be sure that the entire procedure could be executed even on device with limited resources.

Another limitation of Hagggle is that only a single Task at a time is executed, whereas many should be parallelizable. The choice of the group of Tasks that (a) *can* be scheduled together (i.e. their demands on the underlying Connectivity do not conflict), or (b) *should* be scheduled together (i.e. it would be helpful rather than harmful), should be investigated in the future.

The current implementation of the Resource Manager is reactive only, and does not attempt to predict future network connectivity options. To improve the usability of Hagggle we need to provide some level of predictability of the behavior of the opportunistic network.

The current decision process does not take into account monetary costs or energy consumption, although these are key issues in device connectivity today, as they impact

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

battery life and the potentially high cost of “always-on” connectivity. In the future the cost analysis should be expanded to include also these voices.

A further limitation of the Resource Manager is in the mechanism for testing the Internet capability of APs neighbors. Currently Hagggle launches a task to discover the existence of APs. To test if they are Internet capable it creates a Task that, for each of them, checks for Internet connectivity. This task completely lacks a mechanism for selecting APs on the base of configurable criteria that user can specify through a suitable interface. One of these criteria is for example the RSS score. At present Hagggle could try to connect many times to the same AP whose RSS is low and this doesn't make sense if an AP with a better RSS is present.

The current implementation of the Hagggle architecture lacks of a mechanism for defining which DOs can be marked as lowest-priority and can be replaced by higher priority DOs if storage space is an issue (and implement a sort of cache). According to this, mechanisms should be implemented to save storage space (i.e. by compressing data) and caching replacement policies should be applied in the future (i.e. LRU, evict the object that was requested the least recently). These activities should be implemented as Tasks managed by the Resource Manager.


4.1.3 Connectivity manager

Currently the management of the connectivity in Hagggle is not optimized: significant overhead is carried during the switch between infrastructure and ad hoc modes, particularly when switching to infrastructure mode if the DHCP handshake incurs packet loss. For this reason a new best effort way to retrieve a network address has been studied. It uses a particular feature of the DHCP Server to bypass the steps of the DHCP protocol and retrieve the new network address in a faster way. This new technique should be integrated in the next versions.

Currently 802.11 Connectivity in Hagggle doesn't support WPA: this feature should be provided in the next versions.

Another issue that has to be accomplished is the association between WEP Key and BSSID. In the current Hagggle implementation the list of encryption keys (WEP key) is contained in the configuration file. This could prove to be a breach of privacy but, as security and privacy have not been addressed as key concerns in the current version of Hagggle, we don't care about it. However, there's no mechanism to associate a WEP Key to its BSSID: this, could be specified for example in the configuration file.

Tests have been conducted using only 802.11 Connectivity, but laptop and smart phones typically have multiple interfaces: Bluetooth, GPRS/UMTS, infrared. Hagggle's architecture has been designed to take advantage of multiple wireless communication techniques and Hagggle should be able to allow the appropriate Connectivity to be used per-Task, e.g. using GPRS for checking new email existence but waiting for an 802.11 AP to download large email. For this reason, although the GPRS connectivity has not been exploited, as it is not related to the opportunistic vision, in the future it could become a valid backup solution if no connectivity is present, and so it could be useful to provide a mechanism to automatically manage the GPRS connection.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

4.1.4 Forwarding algorithms

Currently Hagggle can support many forwarding algorithm running simultaneously. Two types of algorithm that have been implemented are *DirectForwardingAlgorithm* and *EpidemicForwardingAlgorithm*. In the future it could be useful to integrate forwarding algorithms that are the result of research into opportunistic scenarios. However, iMote data analyses were not advanced enough to provide input to hagggle INFANT implementation.

4.1.5 Web application

All of the downloaded pages and objects are stored as individual WebDO objects, marked by their URL. So, the next time that the HTTP proxy tries to service the same URL, it will find the WebDO. The shortcoming of this mechanism is that the content could not be updated, so in the future, a Task for checking the content of a web page should be done to be sure to use the latest version of it.

4.1.6 Windows mobile specifics

The Hagggle prototype has been developed using Java and targeted initially at Windows XP. Although it's CDC compliant (its target comprises a broad range of consumer and embedded devices like smart communicators, pagers, PDAs) and thus portable to Windows Mobile Platforms, some work has to be done in order to be completely compliant with the Windows Mobile architecture.

In particular, since Windows Mobile 5 (like Windows CE 4.2) doesn't support the same native driver component used in Windows XP to communicate with the NDIS driver, the Native 802.11 interface has to be re-built.


Another issue that has to be investigated is the fact that Windows Mobile 5 doesn't support multihoming (i.e. the management of simultaneous IP connections). In detail, once there are connections on one network, it is not possible to activate the other by binding to the local IP address of that interface and then, connecting out the connection, will fail until all sockets are closed on the other connectivity.

Test should also be conducted to check if the Resource Manager can effectively support the execution of many tasks in parallel as computational resources on PDAs and smart phones (RAM memory in particular) are limited.

4.2 Second Trial

Given the limitation we observed with the iMotes, we decided to go for the following strategy:

- Perform a last iMote experiment where each experimentalist would carry three iMotes: One in query mode, one in listen mode, and one in default hagggle mode. Such an experiment will allow us to study the bias in our previous data sets, and also to understand specificities of Bluetooth technologies that have impacted our observations. After this experiment, iMotes will be made available to colleagues, but we will not use them in the context of Hagggle.
- Migrate to a PDA-based or phone-based platform. We now have enough data on mobility and it is time to migrate to environments where we can test various applications, system limits, forwarding algorithms, and usability of hagggle. The

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	
--	--	---

length of the observation, which was an issue when we started collecting data, is not anymore. The focus is now on testing functionalities and various communication algorithms in real environments.

Recent products announcements (i.e. iPhone by CISCO and Apple, Nokia N800, similar products by Motorola and Samsung) could make the migration easier. However, we do not know yet when these devices will be available, how easy it will be to implement our prototype, and what their real system capabilities will be. We have decided to adopt the following strategy:

1. Create a new task (April07 to November07) in which we will study devices available (mostly PDAs and CellPhones) and make an informed decision to select one (or a small number of different devices) for the rest of the project. In case no device is flexible enough, we will adopt a laptop-based deployment strategy.
2. Delay the second trial to the end of 2007. We have already identified possible environments for the second experiments: MASS in Pisa in September, AINTEC in Bangkok in November, or CoNext in December in NYC (most probable). Therefore, this will delay all deliverable related to the second and third trials by 6 months.
3. Mid 2008, add a new partner that would either be a device manufacturer (e.g. Nokia, Ericsson) or an expert in applications and coding for wireless devices (e.g. www.soft-ocean.com/). This decision will be made before the project second review.

In the meantime, we will keep developing both the windows and the Linux implementations.


The objectives of the second trial will be to test CHILD Hagggle functionalities and communication environment with a small set of real users, possibly using multiple communication technologies. Bluetooth and WiFi are two obvious ones. We would also like to use GSM. But this cannot be committed yet.

Multiple Forwarding algorithms will be deployed in parallel. These algorithms will be defined in WP5.

We will deploy at least one of the following two applications:

- Newsgroup based on experimentalist interest. The application will define a set of topics of interest in the computer networks area, and maybe some local newsgroups so that conference participants can share information about leisure, restaurants, etc.
- Asynchronous messaging. Each experimentalist will be able to send SMS type messages to another experimentalist, and possibly to any Internet user.

CHILD Hagggle should not differ from what has been described in the DoW. Therefore, we refer the reader to the Hagggle DoW for details on CHILD Hagggle.

	Deliverable D3.1: Interim release of Hagggle. Analysis report of the trial and specification of the next trial	 The Hagggle logo features the word "Hagggle" in a white, sans-serif font, centered over a background image of a desert landscape with sand dunes under a blue sky.
--	--	--

5 Conclusion

This deliverable describes the initial experiments performed within the Hagggle project. Instead of a small scale experiment with INFANT Hagggle, we preferred to collect more mobility data and do simple emulations of the INFANT Hagggle. This approach has the following advantages:

- Support other Hagggle research activities with real data, which in turn will allow us to make faster progress on modeling, communities, and forwarding;
- Dissemination: Our traces are used by the research community, and our results have been published and highly referenced;
- Increase the chance of success of the second trial by learning about the current implementations as well as technologies and Operating System limitations.

We believe that the Hagggle project is now in good shape to perform a medium scale experiment in a conference location at the end of 2007. Whilst this is a 6-month delay from the original plan, it gives us time to make an informed platform choice. We are currently discussing whether we should choose a partner with a device manufacturer and adopt a specific hardware platform, or whether we should go for a java implementation on the top of multiple hardware devices.