



HAGGLE 027918

*An innovative Paradigm for Autonomic
Opportunistic Communication*

Specification of first Haggle application and INFANT-Haggle

Deliverable Number: D1.1
Delivery Date: September 2006
Classification: Public
Contact Authors: Haggle partners (Silvia Giordano, Alessandro Puiatti Editors)
Document Version: v1.0 (October 4, 2006)

Contract Start Date: 1 January 2006
Duration: 4 years
Project Coordinator: Thomson, France
Partners: INTEL Research (UK),
EPFL (Switzerland),
Uppsala University (Sweden),
University of Cambridge (UK),
Martel (Switzerland),
Eurecom (France),
CNR (Italy),
SUPSI (Switzerland)

**Project funded by the
European Commission under the
Information Society Technologies
Programme of the 6th Framework
PRIORITY 2
(2002-2006)**



Abstract

In this document, we define the INFANT-Haggle node specification. In order to achieve this, some communication models (e.g. human communication models) are studied, specified and understood. From there on, we derive the core modules of the Haggle node, the elements to be specified to WP2 for realizing the basic communication paradigms, and the elements to be specified to WP3 for rapid experimental evaluation. The INFANT-Haggle release will have no security, will support one platform and one application (messaging).

Contents

1	Introduction	6
1.1	Human based models	6
1.1.1	Human communication models and concept	7
1.1.2	From human communication model to Haggle node architecture	10
1.1.3	From Human Communication process to Haggle Communication process	11
1.2	Mobility Models	13
1.3	Forwarding paradigms for the INFANT-Haggle	13
1.3.1	Taxonomy of forwarding schemes for INFANT-Haggle	13
1.3.2	Network Coding	14
1.3.3	Network Coding in Haggle	18
1.4	Haggle Node Resources Management	20
1.5	Context Management for Haggle node self-* features	20
1.6	Security Requirements	21
2	The INFANT-Haggle Node: from traditional networks to the new network architecture	22
2.1	Pocket Switched Networks	22
2.2	Problems with status quo	24
2.3	A New Set of Mobile Networking Principles	25
2.3.1	Forward using application layer information	26
2.3.2	Asynchronous operation	26
2.3.3	Empower intermediate nodes	26
2.3.4	Message switching	27
2.3.5	All user data kept network-visible	27
2.3.6	Build request-response into the network	28
2.3.7	Exploit all data transfer methods	28
2.3.8	Take advantage of brief connection opportunities	29
2.3.9	Empowered and informed resource management	29
2.3.10	Use and integrate with existing application infrastructure where possible	31
3	The INFANT-Haggle Architecture	31
3.1	Data Manager	32
3.2	Name Manager	33

3.3	Forwarding Manager	34
3.4	Connectivity Manager	35
3.5	Protocol Manager	35
3.6	Resource Manager	35
3.7	Interacting with applications	36
4	Integration and Testbed	37
4.1	Implementation Environment	37
4.2	Haggle Testbed	37
5	INFANT-Haggle Application: Messaging	37
6	Annex I - Acronyms	40
7	Annex II - Code	41

List of Figures

1	Interactive communications model.	8
2	Essential of Human Communication	9
3	Overview of Haggle architecture	10
4	Human communications model.	12
5	Taxonomy of forwarding schemes.	14
6	Current networking architecture for mobile applications	24
7	Overview of Haggle architecture	31
8	Haggle Architecture instantiated for Mail Proxy application	38

1 Introduction

Haggle is a new autonomic networking architecture designed to enable communication in the presence of intermittent network connectivity, which exploits autonomic opportunistic communications (i.e. in particular in the absence of end-to-end communication infrastructure).

We depart from the existing TCP/IP protocol suite, completely eliminating layering above the data-link, and exploiting an application-driven message forwarding, instead of delegating this responsibility to the network layer. To this end, we go beyond already innovative cross-layer approaches, defining a system that uses real best-effort, context aware message forwarding between ubiquitous mobile devices, in order to provide services when connectivity is local and intermittent. We use only functions that are absolutely necessary and common to all services, but that are sufficient to support a large range of current and future applications, more oriented to the human way of communicating (and more in general, the way communities of any type of entities communicate), rather than related to the technological aspect of the communication.

The main components of Haggle are:

1. A revolutionary paradigm for autonomic communication, based on advanced local forwarding and sensitive to realistic human mobility.
2. A simple and powerful architecture oriented to opportunistic message relaying, and based on privacy, authentication, trust, and advanced data handling.
3. An open environment for application and services easy proliferation, thanks to a top down approach that aims to reproduce communities' behavior, which makes Haggle an ideal paradigm for supporting applications with high social and economic impact.

In this document, we completely specify this new networking architecture and implement it for the INFANT-Haggle node. This is the definition of the individual Haggle node, which will further interact with other individuals creating the Haggle community.

This document is mainly focusing on describing the design and implementation aspects of this first individual. However, as there is also a large component of innovative research for the further evolution of the Haggle node, we present in this section the different research areas where we are active and to summarize the directions we are following. For more precise and detailed research discussion on the several topics introduced in this section, we remand to the specific deliverables.

1.1 Human based models

The Haggle node architecture is a specification of algorithms, data structures and protocols that derives from higher level observation, without considering the nodal interaction in Haggle. They are mainly derived by the observation of existing communication models (e.g. in the human world). This because we want to depart from the traditional communication paradigms, which are mainly constrained by technological reasons. Haggle behaves very likely to communities, opportunistically using non-related actions and events (e.g. to meet) for communicating. For this reason we decided to study a some communication models in the nature,

in order to import it in Haggle as a forwarding paradigm. These communication models are intended as the various paradigms through which information is successfully exchanged in real societies. We explored multi-disciplinary literature on communities (e.g. of individuals, of entities, etc) for deriving the way they communicate, from the most primitive and basic, up to the most advanced. In our view, this is the missing link between social, natural and technological science, and can serve as model for long-range scenarios and adaptive explanations for particular behaviors. This will draw the most complex picture of Haggle communicative ability as influenced by several social, human and natural aspects. The models obtained by this first classification should provide a starting point for "store & forward" techniques used to exchange information between Haggle nodes. Therefore, we started from the analysis of the human communications model to better understand and design Haggle architecture.

1.1.1 Human communication models and concept

Communication is a conscious or unconscious, intentional or unintentional process in which feelings and ideas are expressed as verbal and/or nonverbal messages that are sent, received and comprehended. This process can be *accidental* (having non intent), *expressive* (resulting from the emotional state of the person), or *rhetorical* (resulting from specific goals of the communicator). Human communication occurs on the intrapersonal, interpersonal, and public levels. **Intrapersonal communication** is communicating with yourself. It encompasses such activities as thought processing, personal decision making, listening and determining self-concept. **Interpersonal communication** refers to communication that takes place between two or more persons who establish a communicative relationship. Forms of interpersonal communication include face-to-face or mediate conversations, interviews, and small-group discussions. **Public communication** is characterized by a speaker's sending a message to an audience that could be direct, such as face-to-face message delivered by a speaker to an audience, or indirect, such as a message relayed over radio or television.

All communication is dynamic, continuous, irreversible, interactive, and contextual:

- *dynamic*: because the process is constantly in a state of change; as the attitudes, expectations, feelings, and emotions or persons who are communicating change, the nature of their communications changes as well.
- *continuous*: because it never stops. Whether asleep or awake, we are always processing ideas and information through our dreams, thoughts, and expressions; as long as our brain remains active, we are communicating.
- *irreversible*: once we send a message, we cannot undo it.
- *interactive*: we are constantly in contact with other people and with ourselves. Others react to our speech and actions, and we react to our own speech and actions, and then react to those reactions. Thus, a cycle of action and reaction becomes the basis for our communication.
- *contextual*: is the highly complex process of communication because it is a part of our entire human experience. The complexity of the communication dictate that we develop

the awareness and the skills to function effectively as communicators and to adapt to the setting, the people who are present, and the purpose of the communication [2].

In early **models** of theories, the human communication process was seen as linear. According to the *linear* view of communication, the speaker spoke and the listener listened. Communication was seen as proceeding in a relatively straight line. Sooner the linear view was replaced with an *interactional* view, in which the speaker and listener were seen as exchanging turns at speaking and listening, as depicted in figure 1. For example, A spoke while B listened, then B spoke in response while A listened.

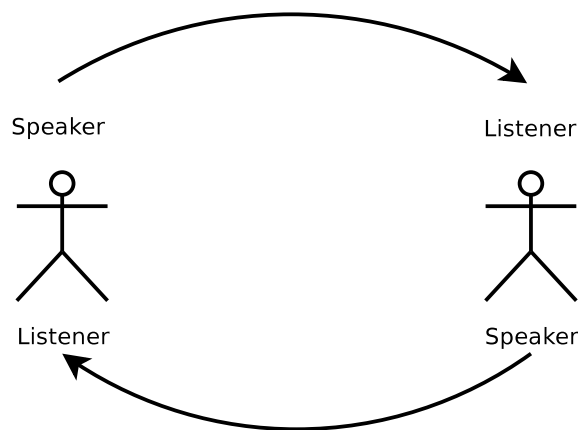


Figure 1: Interactional communications model.

A more satisfying view, the one held currently, sees communication as a *transactional* process in which each person serves simultaneously as speaker and listener: at the same time that you send messages, you are also receiving messages from your own communications and from reactions of the other person[4]. The transactional viewpoint sees each person as both speaker and listener, as simultaneously communicating and receiving messages, and it also sees the elements of communication as interdependent (never independent): each exist in relation to the others.

Regardless of the model that could be used like reference, the process for human communication is not only affected by the speaker and the listener, but there are many other aspects that must be taken in consideration, see figure 2, that are:

- **encoding - decoding:** when a person has to put ideas in speech, he puts them into a code (encoding) that must be understood (decoding) by the target of his speech. There are different types of code and normally more codes are used together in a *layer-fashion* way in which the first code is the necessary background knowledge for the second code and so on. For instance the language (English, Spanish, Indonesian etc..) is the first code for that two people could understand each other, but over that they could use a more technical language (medical, business, lawyer etc.) as the second code.
- **messages:** in human communication, communication messages take many forms and are transmitted or received through one or a combination of sensory organs (channels),

verbally (with words) or non verbally (without words); everything about the speaker communicates a message. It is possible to classify the human messages into four types:

1. **primary** messages: the messages that a person wants to be received by the listener
 2. **metamessages**: a message that refers to another message. It is communication about communication. For example, "Do you understand what I am trying to tell you?" refer to communication and are therefore "metacommunicational"
 3. **feedback** messages: When you send a message (speaking to another person), you also hear yourself. That is, you get feedback from your own message.
 4. **feedforward** messages: is information you provide before sending your primary messages.
- **channel**: is the vehicle or medium through which messages pass. In human communication normally more than one channel may be used simultaneously. In face-to-face conversation, for instance, who speaks also listens (vocal channel), but he also gestures and receives signals visually (visual channel).
 - **noise**: is anything that distorts the message, that prevents the receiver from receiving the message. The noise may be *physical* (others talking loudly, illegible handwriting...), *physiological* (hearing or visual impairment, articulation disorders), *psychological* (preconceived ideas, wandering thoughts), or *semantic* (misunderstood meaning).
 - **context**: communication exists in a context, and that context to a large extent determines the meaning of any verbal or non verbal message. The same words or behaviors may have totally different meanings when they occur in different context [4].

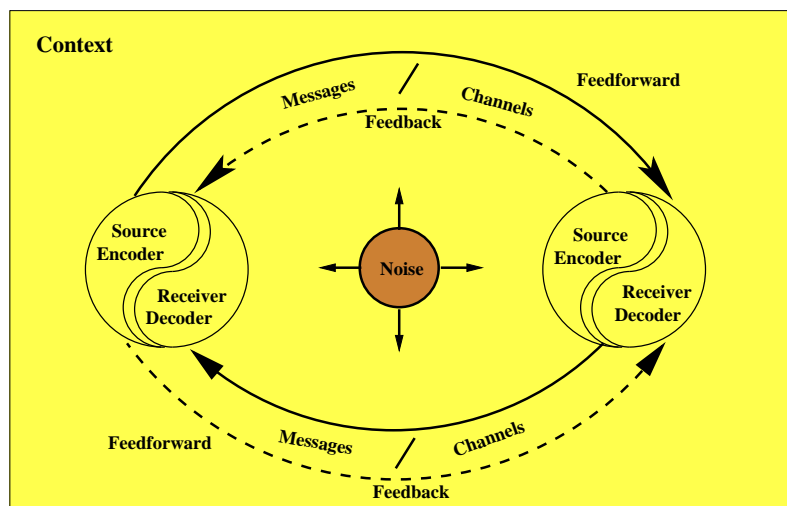


Figure 2: Essential of Human Communication

This is a general model of communication between two people and most accurately depicts interpersonal communication [4]

1.1.2 From human communication model to Hagggle node architecture

Figure 2 depicts general model of the human communication that was used as starting point for designing the INFANT-Hagggle node architecture. The main observation we derived from the analysis of the Human Communication Model (HCM) is that, in any form, the communication between people only functions with the receiver(s) in front of the sender. This being said, the HCM is in complete opposition to the traditional ICT communication model, which is strictly end-to-end. Even if this end-to-end model has been considered an evolution for a long while, it is no more so in the pervasive environment surrounding us (and will have even more). Indeed, this model does not allow benefitting from all the communication opportunities, as in many cases there is no end-to-end connection. Therefore, we designed an Hagggle node architecture oriented to opportunistic message relaying, similarly to the HCM. To reproduce the HCM, Hagggle node has to handle the connectivity, has to be able to forward the messages and to manage the information (as in the memory) and the resource. In order to manage this different tasks we have defined the Hagggle node architecture as shown in Figure 3.

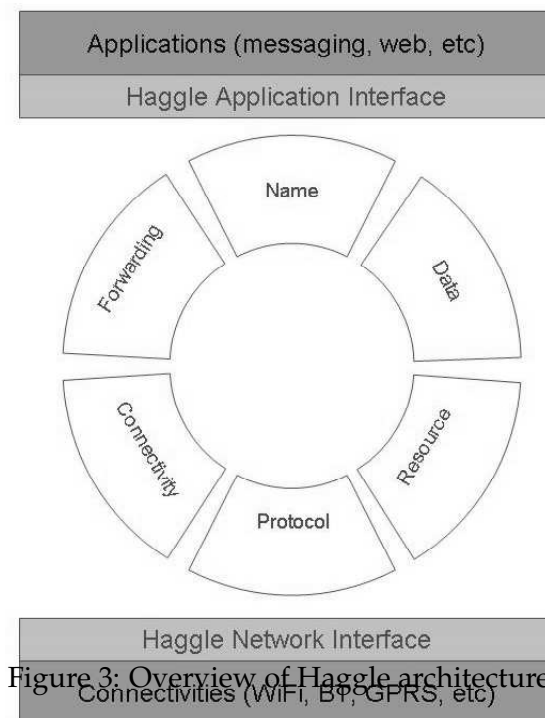


Figure 3: Overview of Hagggle architecture

The tasks are handled by different managers as explained below:

- **Data Manager:** storing, linking together as trees, and searching persistent data objects
- **Name Manager:** organising names of potential destinations for data

- **Forwarding Manager:** deciding suitable next hops for data in transit, interfacing with the application to allow it to ask for data to be sent or solicited, estimating benefits of sending data to nearby names
- **Connectivity Manager:** proposing neighbour discovery tasks, estimation of costs of sending data, providing connections to neighbours on demand
- **Protocol Manager:** using visible neighbours to mark names as nearby, sending and receiving data
- **Resource Manager:** for all tasks proposed, comparison of cost and benefits and decision of which tasks to perform now

A detailed description of the INFANT-Haggle Architecture is presented in Section 3.

1.1.3 From Human Communication process to Haggle Communication process

We can summarize the human communication process as follow (see-Figure 4: the speaker enters in contact with the receiver(s), transmits the message, the listener receives the message (hearing, then listening). During the perception process, the listener tries to understand the meanings of message, evaluate and remember the content. From the information he received, he will have the reasonable response (if any) to the speaker or to others. With this model, a speaker can communicate with any listener that can receive the message, and, if required, the message will be further transmitted by the listener to another listener.

Similarly, in the Haggle CM, the nodes can opportunistically communicate with any other node that can receive the message, and this can be further forwarded, if needed. When each node comes in range of other nodes, it will discover other neighboring nodes. A first communication (exchanging information) takes place. When the sender node wants to send a message to the destination node, based on the information that it received when communicates with other neighbor nodes, it will decide which ones are the best ones to forward the message to (perception process).

Reception and Attention Hearing is not the same as listening. Hearing simply happens when you open the ears or get within the range of some auditory stimulus. Listening, on the other hand, is quite different:

- focus attention on the speaker's verbal and non verbal messages, on what is said and what is not said
- avoid distractions in the environment

In the Haggle model, whenever nodes access the range of others, they can discover and opportunistically exchange the information. The receiving/sending process happens. Only the necessary information will be exchanged. However, even if any node can "hear" a broadcasted message, only the ones really addressed by the sender will receive it and process it. The reception and attention process in Haggle is managed by the *Connectivity Manager* in conjunction

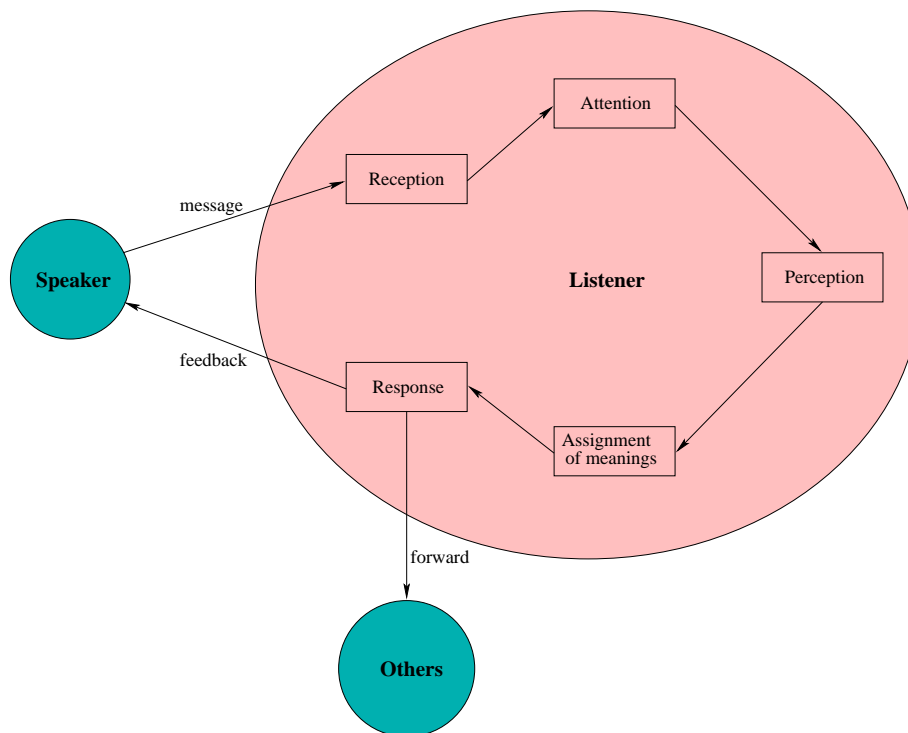


Figure 4: Human communications model.

with the *Resource Manager*. The first one discovers the presence of neighbor nodes and focuses attention on them if the second one has gave it the right to do that.

Perception This process is the most significant part of human and also Hagggle communication. Perceiving is an active process during which a person take the message received and attempts to evaluate the input.

In the Hagggle node model, perception is the process that involves several of the manager interfaces (*Forwarding, Data, Protocol, Resource*). In this process, messages that nodes receive and understand need to be retained for at least some period of time. Each node has its own memory to store and redoes the forwarding process when it meets other nodes and discovers the new suitable candidate to forward the message to. Similarly to the human communication, where participants remember the communication content in order to reproduce it in the future.

Moreover, during the perception process, the persons/nodes evaluate the messages that they received judging which information is meaningful and necessary for them. A similar task is performed in the Hagggle node by the *Resource Manager*.

Assignment of meanings Assignment of meaning occurs when the person decodes the speaker's message, when he understand what the speaker means. This involves, in the Hagggle model, the *Resource Manager*, the *Name Maneger* and the *Data Manager*.

1.2 Mobility Models

The ultimate goal of modeling human mobility is to easily study how to forward messages between Pocket Switched Network (PSN) nodes in a totally distributed way (using models instead of real experiments). Currently we do not have any decent model by lack of understanding of human mobility. The first step is, therefore, to understand the characteristics of human mobility. The modeling activity is currently focused on collecting and analyzing human mobility data. From these data sets, we extract information about forwarding path characteristics. Trying to model too early would be dangerous, as models must capture different aspects of mobility and of communication in PSNs that we have not isolated yet. We approach the modeling work as follows:

- Collect mobility traces in various environments.
- Analyze forwarding paths in all data sets in order to understand the characteristics of human interaction.
- See if existing models, potentially issued from other science domains, can be reused. For example, the theory of small worlds is rich of models and forwarding/routing schemes.
- Analyze how information kept about the nodes met opportunistically can be used to increase the probability to successfully deliver a message in the shortest possible delay. For this, we use information about the frequency of contacts, availability of infrastructure, or node popularity.
- Study social communities. The goal here is to understand if forwarding within a community is more efficient and how these communities are formed. We use data mining and clustering techniques to find these communities and then use the forwarding algorithm emulator to try them on our traces.

All these parallel research works will converge to a set of models that will in turn be used to study forwarding mechanisms and applications for Haggle.

1.3 Forwarding paradigms for the INFANT-Haggle

In opportunistic networks, the concepts of routing and forwarding are mixed together, since routes are actually built while messages are forwarded.

1.3.1 Taxonomy of forwarding schemes for INFANT-Haggle

Figure 5 shows a possible taxonomy of routing/forwarding algorithms in opportunistic networks, that are the basis for designing forwarding protocols for INFANT and next-generations' Haggle. At the bottom of Figure 5 we list relevant examples of each class. More details can be found in [12, 11].

A first classification is between algorithms designed for completely flat ad hoc networks (without infrastructure), and algorithms in which the ad hoc network might also exploit some form

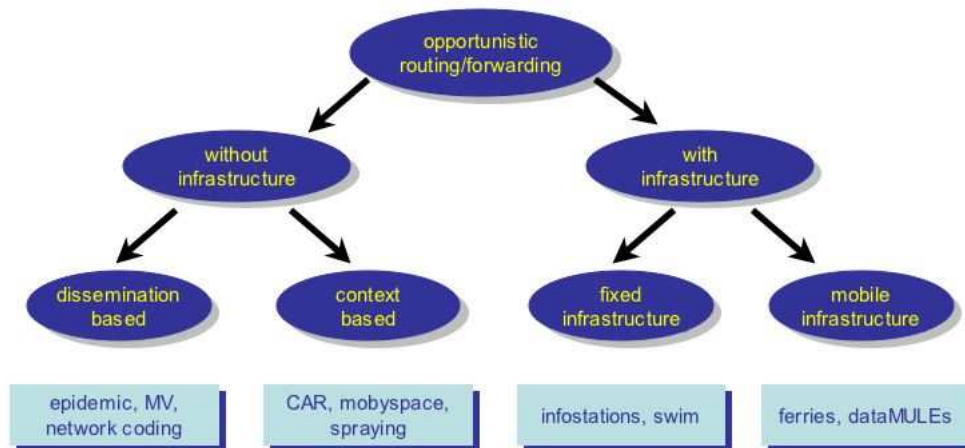


Figure 5: Taxonomy of forwarding schemes.

of infrastructure (e.g., the legacy Internet) to opportunistically forward messages (with infrastructure). In the former case, approaches can be further divided between dissemination based and context based. Dissemination-based algorithms are essentially forms of controlled flooding, and differentiate themselves for the policy used to limit flooding. Context-based approaches usually do not adopt flooding schemes, but use knowledge of the context nodes are operating in to identify the best next hop at each forwarding step. Finally, algorithms exploiting some form of infrastructure can be divided based on the type of infrastructure they rely on. In a first case (fixed infrastructure), the infrastructure is composed by nodes that are more powerful than those commonly present in ad hoc networks, and are located at specific geographical points. These nodes are gateways towards less challenged networks (e.g. gateways can be Wi-Fi Access Points). Quite often, the task of opportunistic routing algorithms falling in this class is to deliver messages to those gateways, since they are supposed to be able to find the eventual destination more easily. In the mobile infrastructure case, nodes of the infrastructure can also move in the space where the ad hoc network is deployed.

1.3.2 Network Coding

Network coding is a new research area that have interesting applications in practical networking systems. With network coding, intermediate nodes may send out packets that are linear combinations of previously received information. There are two main benefits of this approach: potential throughput improvements and a high degree of robustness. Robustness translates into loss resilience and facilitates the design of simple distributed algorithms that perform well, even if decisions are based only on partial information

What is network coding ?

Linear Network Coding

Consider a system that acts as information relay, such as a router, a node in an ad-hoc network, or a node in a peer to peer distribution network. Traditionally, when forwarding an informa-

tion packet destined to some other node, it simply repeats it. With network coding, we allow the node to combine a number of packets it has received or created into one or several outgoing packets. Assume that each packet consists of L bits. When the packets to be combined do not have the same size, the shorter ones are padded with trailing 0s. We can interpret s consecutive bits of a packet as a symbol over the field \mathbb{F}_{2^s} , with each packet consisting of a vector of $\frac{L}{s}$ symbols. With linear network coding, outgoing packets are linear combinations of the original packets, where addition and multiplication are performed over the field \mathbb{F}_{2^s} . The reason for choosing a linear framework is that the algorithms for coding and decoding are well understood. Linear combination is not concatenation: if we linearly combine packets of length L , the resulting encoded packet also has size L . In contrast to concatenation, each encoded packet contains only a fraction of the information contained in original packets. One can think of linear network coding as a form of information spreading.

Encoding:

Assume that a number of original packets M_1, \dots, M_n are generated by one or several sources. In linear network coding, each packet through the network is associated with a sequence of coefficients g_1, \dots, g_n in \mathbb{F}_{2^s} and is equal to $X_k = \sum_{i=1}^n g_i M_{ik}$, where M_{ik} and X_k is the k^{th} symbol of M_i and X respectively.

For simplicity, we assume that a packet contains both the coefficients $g = (g_1, \dots, g_n)$, called encoding vector, and the encoded data $X_k = \sum_{i=1}^n g_i M_{ik}$, called information vector. The encoding vector is used by recipients to decode the data, as explained later.

Encoding can be performed recursively, namely, to already encoded packets. Consider a node that has received and stored a set $(g_1, X_1), \dots, (g_m, X_m)$ of encoded packets, where g_j [resp. X_j] is the encoding [resp. information] vector of the j^{th} packet. This node may generate a new encoded packet (g', X') by picking a set of coefficients h_1, \dots, h_m and computing the linear combination $X' = \sum_{j=1}^m h_j X_j$. The corresponding encoding vector g' is not simply equal to h , since the coefficients are with respect to the original packets M_1, \dots, M_n ; in contrast, straightforward algebra shows that it is given by $g' = \sum_{j=1}^m h_j g_j^i$. This operation may be repeated at several nodes in the network.

Decoding :

Assume a node has received the set $(g_1, X_1), \dots, (g_m, X_m)$. In order to retrieve the original packets, it needs to solve the system $\{X_j = \sum_{i=1}^n g_j^i M_i\}$ (where the unknowns are M_i). This is a linear system with m equations and n unknowns. We need $m \geq n$ to have a chance of recovering all data, i.e. the number of received packets needs to be at least as large as the number of original packets. Conversely, the condition $m \geq n$ is not sufficient, as some of the combinations might be linearly dependent. However, and this is a major appeal of network coding, this is easy, as we discuss next.

How to Select the Linear Combinations

The problem of network code design is to select what linear combinations each node of the network performs. A simple algorithm is to have each node in the network select uniformly at random the coefficients over the field \mathbb{F}_{2^s} , in a completely independent and decentralized manner. With random network coding there is a certain probability of selecting linearly dependent combination. This probability is related to the field size 2^s . Simulation results indicate that even for small field sizes (for example, $s = 8$) this probability becomes negligible. Alternatively, we

can use deterministic algorithms to design network codes. The polynomial-time algorithm for multicasting in, sequentially examines each node of the network, and decides what linear combinations each node performs. Since each node uses fixed linear coefficients, the packets only need to carry the information vector. There also exist deterministic decentralized algorithms that apply to restricted families of network configurations.

Practical Considerations

Decoding: Decoding requires solving a set of linear equations. In practice, this can be done as follows. A node stores the encoded vectors it receives as well as its own original packets, row by row, in a so-called decoding matrix. Initially, it contains only the non-encoded packets issued by this node with the corresponding encoding vectors (if any, else it is empty). When an encoded packet is received, it is inserted as last row into the decoding matrix. The matrix is transformed to triangular matrix using Gaussian elimination. A received packet is called innovative if it increases the rank of the matrix. If a packet is non-innovative, it is reduced to a row of 0s by Gaussian elimination and is ignored. As soon as the matrix contains a row of the form e_i , this node knows that x is equal to the original packet M_i . This occurs at the latest when n linearly independent encoded vectors are received. Note that decoding does not need to be performed at all nodes of the network, but only at the receivers.

Generations: For all practical purposes, the size of the matrices with which network coding operates has to be limited. This is straightforward to achieve for deterministic network codes but more difficult with random network coding. For the latter, packets are usually grouped into so called generations, and only packets of the same generation can be combined. Size and composition of generations may have significant impact on the performance of network coding. Similar considerations hold for the size of the finite field. Both parameters allow to trade off performance for lower memory requirements and reduced computational complexity.

Delay: The fact that packets need to be decoded has a minor impact on delay. It is usually not necessary to receive all encoded packets before some of the packets can be decoded (i.e., whenever Gaussian elimination leads to a row in the form (e_i, M_i)). Together with a reduction in the number of required transmissions, the overall end-to-end delay with network coding is usually not larger than the normal end-to-end delay in realistic settings.

Finite field operations: Network coding requires operations in \mathbb{F}_{2^s} , i.e., operations on strings of s bits which are somewhat different from usual addition and multiplication. If s is small (e.g. $s = 8$), a faster alternative is to use discrete logarithms.

What are the benefits of network coding ?

Theoretically proven results about network coding mainly concern performance improvements in static settings. We review these first and then discuss random distributed settings.

Throughput Gain in Static Environment: A primary result that sparked the interest in network coding is that it can increase the capacity of a network for multicast flows. More specifically, consider a network that can be represented as a directed graph (typically, this is a wired network). The vertices of the graph correspond to terminals, and the edges of the graph corresponds to channels. Assume that we have M sources, each sending information at some given rate, and N receivers. All receivers are interested in receiving all sources.

Theorem 1. Assume that the source rates are such that, without network coding, the network can support each receiver in isolation (i.e. each receiver can decode all sources when it is the only receiver in the

network). With an appropriate choice of linear coding coefficients, the network can support all receivers simultaneously.

In other words, when the N receivers share the network resources, each of them can receive the maximum rate it could hope to receive, even if it were using all the network resources by itself. Thus, network coding can help to better share the available network resources. Network coding may offer throughput benefits not only for multicast flows, but also for other traffic patterns, such as unicast.

An interesting point is that network coding allows to achieve the optimal throughput when multicasting using polynomial time algorithms. In contrast, achieving the optimal throughput with routing is NP-complete. Thus, even when the expected throughput benefits of network coding are not large, we expect to be able to achieve them using simpler algorithms. We expand on this point in the following.

Robustness and Adaptability : The most compelling benefits of network coding might be in terms of robustness and adaptability. Intuitively, we can think that network coding, similarly to traditional coding, takes information packets and produces encoded packets, where each encoded packet is equally important. Provided we receive a sufficient number of encoded packets, no matter which, we are able to decode. The new twist that network coding brings, is that the linear combining is performed opportunistically over the network, not only at the source node, and thus it is well suited for the (typical) cases where nodes only have incomplete information about the global network state.

Let's consider networks where packets may be dropped. We argue that for some applications the benefits of network coding outweigh those of the approaches employed today. Forward error correcting (FEC) schemes at a packet level offer an alternative to automatic repeat request (ARQ), for applications with high load and low delay requirements. Recently, we have witnessed the emergence of Fountain codes, a set of rate-less codes particularly suited for such applications. Fountain codes are still end-to-end: packets are encoded at the source and decoded at the destination, while intermediate nodes are only allowed to replicate and forward packets. Applying ideas from network coding in this context, i.e. allowing intermediate nodes to also form linear combinations, can lead to significant improvements. For example, consider a source A that would like to transmit information to a destination C. On the path from A to C there exists a router B that can perform network coding operations. Assume that node A sends encoded packets, that are dropped on paths AB and BC with probability ϵ_{AB} and ϵ_{BC} respectively. Using an end-to-end FEC scheme, i.e. having the destination C decode the packets it receives, restricts the rate to $R1 \leq (1 - \epsilon_{AB})(1 - \epsilon_{BC})$. If we allow the router B to perfectly decode and re-encode, we will achieve the optimal min-cut rate $R2 \leq \min\{(1 - \epsilon_{AB}), (1 - \epsilon_{BC})\}$, but at the cost of additional delay: we have to wait at node B to receive sufficient encoded packets to be able to decode and re-encode the information. Using an ARQ scheme will again allow to achieve rate R2, but again at the cost of increased delay.

Alternatively, node B can at each time instance form and send random linear combinations of the encoded packets it has received up to that time, without waiting for all encoded packets. We can then achieve the optimal rate R1 without an additional delay. Moreover, it is sufficient to perform xor operations. This scheme in its full generality can be applied over an arbitrary network topology, and with diverse traffic load (multicasting, unicasting, broadcasting, etc.)

1.3.3 Network Coding in Hagggle

Network coding impact is potentially very high in Hagggle, as it can be very beneficial at several levels, ranging from P2P distribution to path solution in pervasive environment. NC will take place in the Forwarding Manager

P2P File Distribution: Probably the most widely known application using network coding is Avalanche developed by Microsoft. Generally, in a peer-to-peer content distribution network, a server splits a large file into a number of blocks. Peer nodes try to retrieve the original file by downloading blocks from the server but also distributing downloaded blocks among them. To this end, peers maintain connections to a limited number of neighboring peers (randomly selected among the set of peers) with which they exchange blocks. In Avalanche, the blocks sent out by the server are random linear combinations of all original blocks. Similarly, peers send out random linear combinations of all the blocks available to them. A node can either determine how many innovative blocks it can transmit to a neighbor by comparing its own and the neighbors matrix of decoding coefficients, or it can simply transmit coded blocks until the neighbor receives the first non-innovative block. The node then stops transmitting to this neighbor until it receives further innovative blocks from other nodes. Coding coefficients are transmitted together with the blocks, but since blocks usually have a size of hundreds of kilobytes, this overhead is negligible. Network coding helps in several respects. 1) It minimizes download times; in a such large scale distributed peer-to-peer system, optimal packet scheduling is very complex, particularly if the participating hosts only have very limited information about the underlying network topology. With network coding, the performance of the system depends much less on the specific overlay topology and schedule. Consequently, very simple mechanisms that construct a random overlay can be used. The authors show that network coding outperforms traditional forwarding or FEC based peer-to-peer systems by a significant margin. 2) Due to the diversity of the coded blocks, a network coding based solution is much more robust in case the server leaves early (before all peers have finished their download) or in the face of high churn rates (where nodes only join for a short period of time or leave immediately after finishing their download). 3) In contrast to forwarding based protocols, their network coding protocol suffers only a small performance penalty when incentive mechanisms to cooperate are implemented (e.g. tit-for-tat to prevent free-riding).

Wireless Networks: Bidirectional traffic in a wireless network: as explained previously, network coding can improve throughput when two wireless nodes communicate via a common base-station. This setting can be extended to the case of multi-hop routing in a wireless network (or any other network with physical layer broadcast) where the traffic between the two end nodes is bidirectional and both nodes have a similar number of packets to exchange. Given a schedule that alternates between adjacent routers, after a few initial steps all intermediate routers have packets buffered for transmission in both directions of the path. Whenever a transmission opportunity arises, a router combines two packets, one for each direction, with a simple xor and broadcasts it to its neighbors. Both receiving routers already know one of the packet in the broadcast is coded over, while the other packet is new. Thus, each broadcast allows two routers to receive a new packet, effectively doubling the capacity of the path. Overhearing a packet of a neighbor that is coded over information previously forwarded to the neighbor serves as a passive acknowledgment. This allows to make better use of transmission opportunities at routers that only have new packets buffered for a single direction. In this case,

one of the new packets is combined with an old packet for the reverse direction for which no passive acknowledgment has been received.

Residential wireless mesh networks: Even a limited form of network coding which only uses xor to combine packets may significantly improve network performance in wireless mesh networks. All transmissions are broadcast and are overheard by the neighbors. Packets are annotated with summary information about all other packets a node already heard. This way, information about which nodes hold which packets is distributed within the neighborhood. A node can xor multiple packets for different neighbors and send them in a single transmission, if each neighbor already has the remaining information to decode the packet. In experiments with 802.11 hardware, the authors show that their mechanism almost doubles network throughput.

Many-to-many broadcast: Network-wide broadcast is used for a number of purposes in ad-hoc networks (e.g., route discovery) and can be implemented much more efficiently with network coding [15, 11]. Already a simple distributed algorithm for random network coding reduces the number of transmission by a factor of 2 or more, leading to significant energy savings. In such a setting, a larger transmit power directly translates into a reduction in the number of required transmissions, which allows for interesting energy tradeoffs. Energy expenditure is either evenly distributed among the nodes or covered by only a few nodes (maybe with longer battery life). There is a larger flexibility in the distribution the energy requirements compared to conventional algorithms. Algorithms for challenging wireless networks which may be very sparse or highly mobile are investigated in [28]. Usually, algorithms for such networks employ some form of intelligent flooding to combat the adverse network conditions. With network coding, performance improvements are much larger than the ones reported above, indicating that network coding is particularly suitable when robust operation is of high importance. The authors present a simple mechanism for network wide broadcast that has a much lower overhead than flooding and also discuss practical implementation issues for network coding in wireless networks. Similar performance benefits can be expected when nodes spend most of their time in sleep mode to save energy or use randomized beamforming with directional antennas. All of these cases have in common that a specific pair of nodes is not likely to be able to communicate at a given time.

Ad-hoc Sensor Networks: Untuned radios in sensor networks: A novel and interesting application for network coding is, to use it to cope with untuned transceivers. For sensor nodes which should be as simple and cheap to manufacture as possible, the quartz oscillator to tune the radio to a specific frequency makes up a significant fraction of hardware cost and design complexity. The authors propose to replace the analog oscillator by a much simpler on-chip resonator. As a consequence, the transceivers radio frequencies depend to some degree on variations in the manufacturing process and two such devices are not very likely to be able to communicate directly. However, in dense sensor networks with thousands of tiny devices and multiple radios per device, a multi-hop path between information source and data sink will most probably exist. With random network coding, it is possible to use these paths without having to explicitly find them, and without the excessive overhead of flooding. In their paper, the authors do not propose a specific a protocol but rather give a theoretical analysis, that, given their assumptions, untuned radios with network coding perform only a factor of $\frac{1}{e}$ worse than perfectly tuned radios without network coding. They conclude, that operating networks in such a randomized fashion may be preferable to the more traditional way of controlled operation, since it implies a simpler radio architecture.

1.4 Hagggle Node Resources Management

In the Hagggle architecture the Resource Manager has a central role. Each other Manager has to submit tasks to the Resource Manager, specifying the benefit of executing that particular task. The Resource Manager evaluates the cost related to each submitted task, and decides which task to execute at what time.

For example, a critical interaction takes place between the Forwarding Manager (where forwarding algorithms reside), and the Resource Manager. The Forwarding Manager proposes tasks, specifying, from its standpoint, the benefit of forwarding a packet via a particular next hop. The Resource Manager evaluates the overall benefit of the proposed task (also including, for example, the benefit for the particular application generating the packet), and the associated cost (e.g., the energy cost of the task, the money spent transmitting the packet, ...). Based on utility functions applied to costs and benefits, the Resource Manager ranks tasks, and ultimately decides when to actually send packets.

Among the resources contributing to task costs, energy and memory are of particular interest. Energy requirements of future mobile devices is going to steadily increase, while the improvements in battery capacity does not seem able to keep the pace [10]. At the same time, alternative sources of energy (e.g., harvesting and scavenging from the environment) are not expected to be able to replace batteries for a long time [13]. Thus, techniques to conserve energy in mobile wireless systems play a critical role. Particularly related to the Hagggle scenario are energy-management solutions devised for ad hoc and sensor networks [1]. These techniques will be adapted to the Hagggle disconnected and opportunistic environment. Memory management will also be critical. On one hand, caching and data-dissemination techniques spread information in the network thus improving data availability and performance in terms of delay. On the other hand, filters are required to select the valuable information for the Hagggle node, out of the huge information space available in the environment. Therefore, we are investigating algorithms to decide what information to store at each node. This is actually a trade off between the expected utility of the information, and the cost of storing it in the local memory. Also in this case, techniques designed for ad hoc networks are a fundamental reference [14]. We are also improving and adapting to the Hagggle scenario results on data dissemination in partially disconnected networks [8].

1.5 Context Management for Hagggle node self-* features

Contextual information is at the core of the autonomic features of the Hagggle node. It is imperative to identify which information should compose this knowledge base, how the Hagggle node should learn the behavior of the user and of other nodes around it, how it should acquire memory and forget. This actually shares some concept with memory management described before. However, context management is interested in those subset of data passing through the node that makes the node aware of the environment it is operating in. Context-based forwarding algorithms are one of the main functionalities that exploit such context information. When an Hagggle node is unable to forward directly a message to the destination, it must decide where to forward the request next. In literature, there are many and many strategies for selecting neighbors, ranging from the most basic one: flooding, up to very sophisticated strategies that uses statistical and/or artificial intelligence strategies for improving the selection. In Hag-

gle, which is based on the assumption of a pervasive intelligent environment, a very natural strategy is the context based one. In this case, the context is used to identify the suitability of encountered nodes to carry data (closer) to the eventual destination(s).

Context at each node is build based on a number of elements, including the node identity, the identity of current neighbors, the history of encountered neighbors, the node's user preferences, etc. Context management has to filter this huge amount of information, discard redundant and stale data, encode relevant data with numerical indices that can be used, for example, by context-based forwarding modules.

For now, the context of a node is defined as its identity, the identity of its current neighbors, and the history of attributes of nodes encountered in the past. The node identity is a rich set of attributes describing the node's user in the current environment. It may include, for example, a name, an e-mail address, an ordinary address, the organization the user is working for, etc. By exchanging (portions of) these identities, nodes become aware of each other during direct contacts. Finally, the history is a filtered aggregation of attributes collected during identity exchanges. One such aggregate is the probability of encountering a particular attribute. Specifically, for each attribute seen in neighbors identities, an encounter probability is computed. The probability increases when the same attribute is met again (possibly, during contacts with different nodes), and decreases otherwise. This is an example of a simple - yet efficient - way of encoding historical context information in an easy-to-use numerical index, which is exactly what other Haggle functionalities (e.g., forwarding) require.

We are also planning to use automatic learning techniques (such as stochastic learning, fuzzy networks, bayesian networks) to improve the efficiency of nodes in learning the context they are operating in.

1.6 Security Requirements

INFANT-Haggle node will have no security, but we already started to investigate the specific security requirements.

Haggle nodes form a temporary network without the help of any infrastructure and without an a priori trust relationship between them. Existing trust establishment mechanisms that require the availability of a centralized trusted authority are not suitable for such communications. A concept that nicely fits with the underlying opportunistic networking model is offered by optimistic security protocols whereby some communication with trusted third parties might be required but the correctness of the security protocol does not require on-line connectivity. Furthermore, Haggle nodes are assumed to have limited resources (e.g. memory, battery). Scarcity of resources would inherently foster nodes to selfishly behave in order to optimize the usage of their resources. Yet, such behavior can have a strong impact on the performance of the network. In order to reduce the effect of selfishness, collaboration among parties must be encouraged by incentive mechanisms. Such cooperation enforcement schemes would guarantee fair and efficient networking operations.

Guaranteeing fairness in the community however does not protect against attacks aiming at corrupting the integrity and confidentiality of messages being forwarded between peers. In order to prevent such attacks, basic security services that provide confidentiality, integrity and key management need to be redefined in the context of the Haggle network. For example,

forwarding requests and messages without accessing their content is a hard problem that relies on solutions for computing with encrypted functions and homomorphic encryption. Design of practical solutions amenable to applications such as opportunistic networking is an open problem that calls for a judicious blending between protocol complexity and realistic design choices imposed by the underlying communication model. Moreover, network coding schemes must be protected against Denial of Service (DoS) attacks such as bogus injection or tampering, with some new integrity mechanisms that have to be adapted to the specific environment.

All security requirements that have been so far defined in this section strongly depend on a common problem that is key management. Since there is a lack of organization, key management should not rely on some designated security server. Key pre-distribution protocols are good candidates for this problem and should be analyzed in the context of the Haggie network.

This security requirements will be taken into account in future versions of the Haggie node and communication architecture.

2 The INFANT-Haggie Node: from traditional networks to the new network architecture

The core point of this deliverable is the description of the INFANT-Haggie node architecture. This architecture was derived from the initially analysis we conducted and do not include all the research we briefly discussed in previous sections (e.g. Network Coding, security), which will be part of farther releases of Haggie node architectures.

2.1 Pocket Switched Networks

In designing a new network architecture, it is first important to define the scenario in which that architecture will be used. IP, for example, was designed against a backdrop of a multitude of existing networks, and with the primary needs being resilient end-to-end communications in the presence of node failures, as befits its originator, the US Department of Defense [3].

PSN is the term we use to describe the situation faced by today's mobile information user. Such users have one or more devices, some/all of which may be with them at any time, and they move between locations as part of a normal schedule. In so moving, the users can spend some (or much) of their time in "islands of connectivity", i.e. places where they have access to infrastructure such as 802.11 access points (APs) which they can use to communicate with other nodes via the Internet. They also occasionally move within wireless range of other devices (either stationary or carried by other users) and are able to exchange data directly with those devices.

Thus, in PSN, there are three methods by which data can be transferred, namely neighborhood connectivity to other local devices, infrastructure connectivity to the global Internet, and user mobility which can physically carry data from place to place. For the former two methods, the connectivity is subject to a number of characteristics, including those of bandwidth, latency, congestion, synchronicity (e.g. email or SMS are asynchronous, while ad-hoc 802.11 is synchronous), the duration of the transfer opportunity (i.e. the time till the device moves out

of range), and also monetary cost (usually only for infrastructure). For the latter method of user mobility, users acting as “data mules” can transfer significant amounts of data, and while users’ movements cannot in general be controlled, they can be measured [6], and patterns in those movements can be exploited.

In addition to the issue of network connectivity, we must also consider the usage model for PSN. While different applications have different network demands, we can distinguish particular broad classes which are known to be useful: (a) *known-sender* where one node needs to transfer data to a user-defined destination. The destination may be another user (who may own many nodes), all users in a certain place, users with a certain role (e.g. “police”), etc. The key point is that, often, the destination is not a single node but is instead a set of nodes with some relationship, e.g. the set of nodes belonging to a message recipient. (b) *known-recipient* in which a device requires data of some sort, e.g. the current news. The source for this data can be any node which is reachable using any of the three connectivity types, including via infrastructure (e.g. a news webpage), neighbours (e.g. a recent cache of a news webpage) or mobility (e.g. the arrival of a mobile node carrying suitable data). In both classes described above, the endpoints of a network operation are no longer described by network-layer addresses, but are instead a set of desirable properties. As a result, general network operations no longer have single source and destination nodes.

Finally, in PSN situations, resource management is a key issue. Mobile devices have limited resources in terms of storage, network bandwidth, processing power, memory, and battery. The latter is perhaps the most important, since the others can potentially be reclaimed without the user’s assistance, while charging the battery requires the user to perform the physical act of plugging it in, and restricts the device’s mobility while charging. Other resources are also precious, particularly in the face of demands imposed by the usage scenarios above, where devices may need to use storage and network bandwidth to help forward messages for other devices. However, there is also much cause for optimism — storage capacities are increasing exponentially, wireless networking has the useful property of spatial reuse, and processing power on mobile devices is growing with Moore’s Law. For power, many devices are plugged in more often than not, e.g. notebook computers, and low power electronics allows current mobile phones to last for many days on a single charge.

From the discussion above, we extract three motivations for a networking architecture in the PSN environment, in order of importance:

- Allow applications to take advantage of all types of data transfer (neighborhood, infrastructure, mobility) without having to specifically code for each circumstance.
- Allowing networking endpoints to be specified by user-level naming schemes rather than node-specific network addresses, thus each network operation can potentially involve many endpoints.
- Allowing limited resources to be used efficiently by mobile devices, taking into account user-level priorities for tasks.

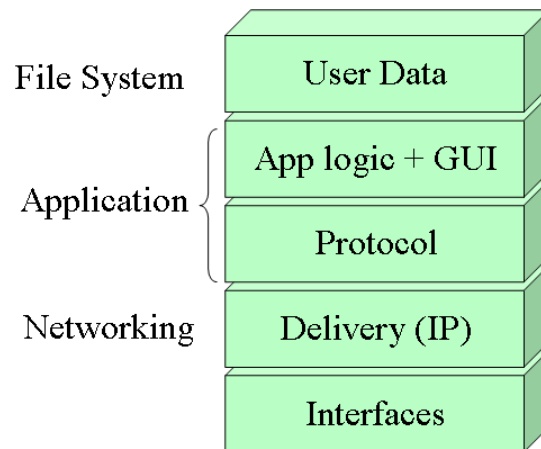


Figure 6: Current networking architecture for mobile applications

2.2 Problems with status quo

Current applications perform very badly in the PSN environment, since they are typically designed around some form of infrastructure which is not always available. While some applications can cope with infrastructure blackout, e.g. with a “disconnected” or “offline” mode, most do not. Direct, neighbourhood connectivity is used by very few widely used applications, and human mobility is deliberately used by almost none. Thus, when infrastructure is not present, users are presented with huge inconveniences since the applications which are familiar to them stop working, and are forced to take on the task of understanding these situations so that they can be productive despite this application failure. For instance, users may require many alternative applications in order to do a single task depending on the situation, e.g. a file can be exchanged by either email, or by putting it on a website for download, or by using an instant messaging client, or by direct Bluetooth or infrared transfer. More likely, the user will simply invest in a USB key and manually bypass the huge inconvenience of the status quo.

The root cause of this is the fact that applications are provided with a networking interface that only understands streams of data directed at anonymous numeric endpoints (namely TCP/IP). As illustrated in Figure 6, this forces developers to implement protocols for naming, addressing and data formatting internally in the applications themselves, e.g. SMTP, IMAP and HTTP. While at the GUI level, applications have general user-level tasks such as “send this file to James,” once a particular network protocol such as SMTP is imposed on that task, it becomes the a more specific task, e.g. “send this file to the server pointed at by the MX record in the DNS record of the domain name part of james.w.scott@intel.com”. The latter task is specific to a particular kind of connectivity scenario, in this case infrastructure-based. It is therefore impossible to execute even if James’s device is in the neighbourhood at that time — i.e. even if the user-level task could be satisfied.

Another problem with the current networking API is that it is synchronous. Applications cannot indicate a network task to be performed and then exit, since finished applications have all

their TCP/IP sockets closed. For example, an email application with pending outgoing email in the outbox will not be able to use a passing AP to send this email if the application is not running when the AP is passed. Therefore, an application in the PSN environment has to be constantly on and monitoring the connectivity status of the device. This increases the complexity of a disconnection-aware application, since it must be able to wait through periods of bad connectivity and detect and perform networking actions when a suitable endpoint is again visible. It also increases the load on mobile device resources, since many applications would have to be present in the background at all times.

A third problem is that persistent user data is kept by applications in a file system which, in the current node architecture, is unconnected from the networking system (again illustrated in Figure 6). This means that all “sharing” of data between nodes must often be conducted by applications themselves¹. The biggest example of this is the device synchronisation problem — when a user has multiple devices, they must explicitly run an application on each which pulls their data out of the file system and shares it with their other device(s). Such synchronisation is often a source of much inconvenience for users, since the sync tools must understand the different ways that each user application uses the file system to store data and metadata, and often has to translate it so that different applications can be sync’d with the same data. Another example is in distributed web caching — the exact web page that a user wants may be in the cache of a neighbouring node, but since web browsers do not explicitly support the transfer, there is no way to get this off the neighbour’s file system and into the network to be shared with the user.

The final problem identified is that applications have no easy way to prioritise the use of a mobile devices’ limited resources. These resources include persistent storage, network bandwidth, and battery energy. Currently, an application such as a web browser must estimate by itself how much of the storage can be used for non-critical history caching, or how much network bandwidth should be used for pre-fetching of web pages. This decision is often passed on to the user, who might have to adjust settings manually, at the application level (e.g. “how much disk to use as cache”), at the hardware level (e.g. turning on or off wireless network interfaces depending on the battery level), or by only running certain apps when they do not want to prioritise network bandwidth for other tasks (e.g. network-hungry file-sharing apps). These controls are coarse at best, and require expert understanding in order to properly exercise them. The result is that resources are often used inefficiently.

2.3 A New Set of Mobile Networking Principles

We now present a set of interrelated principles which we believe are fundamental implications of the situation faced by users’ devices in the PSN environment, and can provide solutions to the problems with the status quo. These guide the design of the Haggle architecture presented below, but are also applicable to other architectures for any networking scenario with similar characteristics. Note that we do not claim that the individual principles below are novel, some (such as message switching) are very well-known. We do believe, however, that we are the first to assemble this particular *set* of principles.

¹Networked file systems can be used for data sharing, but these rely heavily on good connectivity, often to a particular server, and as such are not generally usable in PSN

2.3.1 Forward using application layer information

Applications should not be forced to specify endpoints using addresses, such as IP addresses, that are meaningless at the user level. Instead, endpoints should be specified using higher-layer information, e.g. the URL of a website. By performing forwarding with such information, Huggle can satisfy the application's needs using any form of connectivity, e.g. going to that URL directly if there is infrastructure, or obtaining it directly from a neighbour who has a cached copy, or using a node known to be passing an AP soon to physically carry the request and propagate the answer back, etc. In other words, we need to move from *node-centric* networking to *data-centric* networking.

Taking this example a step further, website URLs are often found using search engines, whereas the user's request is actually for information matching a particular set of keywords. Huggle can use such keywords directly, e.g. a request for "current world news" can be satisfied with cached copies in the environment of any news website (perhaps with a user-specified order of preference, or a whitelist/blacklist).

Similarly, using an email address for forwarding restricts a messaging application to using email protocols and infrastructure, while using a phone number restricts the application to forwarding using SMS. By allowing Huggle to use the person's name (the higher-level, more meaningful identifier), it can use any protocol for which it has a mapping between the high-level name and a protocol-meaningful address.

2.3.2 Asynchronous operation

Asynchronicity is important in three ways in Huggle. First, applications should be able to indicate networking actions asynchronously from the actions taking place. This is in contrast to the current model where applications must be "on" throughout the transmission (as described in Section 2.2), and thereby reduces the complexity of a PSN-friendly application. Second, this also means that the decision of precisely which next-hop node(s) to forward data to can be left as late as possible; in other words, the forwarding algorithm can use "late binding" when assigning a low-layer next hop address, allowing it to best utilise up-to-date local context information about which next-hop nodes allow the data to make the most progress toward a destination. Third, asynchronicity is key to the store-and-forward nature of Huggle, which allows it to cope with non-contemporaneous connectivity between endpoints in a way that end-to-end protocols such as TCP cannot.

2.3.3 Empower intermediate nodes

In PSN, intermediate nodes (i.e. nodes on the transmission path that are not specified as destinations in the data being transferred) may also be valid destinations for data. For example, if a mobile device acts as a forwarding node for a webpage, that device may wish to keep a cache of the webpage in case its own user later wishes to view it, or it comes into contact with another device which requires that information. This is in contrast to infrastructure-based networking where intermediate nodes do not usually reconstruct application-layer data to decide whether it is locally useful, and the data is simply transmitted end-to-end.

This is effectively ad-hoc multicasting, where the multicast group can be joined at any time by any device which can see (or get to) a copy of the data. It has significant advantages over demand-driven data transmission, since a demand for a data item at a particular location (with no infrastructure) cannot be transmitted easily to a node which is moving towards that location and has a chance to pick that data up. However, if that node opportunistically stores the data, perhaps using policies or learning algorithms to determine whether it is likely to be “popular”, then the data can arrive unbidden at a location where it is useful.

2.3.4 Message switching

All of the above three principles imply that message switching is more suitable than packet switching for Haggle. This is not to say that the underlying networks might not use packet switching, but that full application-level messages should be exchanged by neighbouring Haggle nodes when possible. Message switching means that application-layer forwarding information does not have to be duplicated across many packets, it facilitates asynchronous operation by the networking subsystem, and it means that intermediate nodes are provided with the whole message so they can act as destinations as well as forwarding points for any given message.

2.3.5 All user data kept network-visible

Asynchronicity implies that user data in transit needs to be kept in a node’s Haggle framework. However, we take this principle further: In PSN, *all* user data should be made visible to Haggle at all times. In addition, data must be marked with metadata about its user-level properties, such as access authorisation, creation/modification/expiry times, etc. We have two main reasons for this.

First, a significant fraction of user data is inherently shared, i.e. the user’s task involves making it available to other users according to some access control profile. For example, CVS file stores are shared by many users via an infrastructure-based communication model. By making all user data visible to Haggle, such data can be transmitted to other authorised users without relying on infrastructure, making CVS-like applications capable of running under general network conditions. Note that we do not tackle the general data merging/versioning problems that CVS does, but that we simply provide a means for the communications part to be abstracted.

Second, users often have more than one device. Therefore, even a user’s most private data should be network-visible, if only for transmission to other devices that they own (or devices that they trust, e.g. an Internet-based backup service). Currently, data synchronisation between multiple user devices is a very thorny problem both for the developers of such tools, and for users who have to manually associate devices that they want synchronised. We can alleviate some part of this problem by making sure all user data is visible to Haggle and marked with information on who is allowed to access it.

By making all user data visible to the network, we decouple the data from particular nodes and allow it to flow to the set of nodes with a valid interest in it. With Haggle, we aim to achieve this in the face of flexible connectivity environment inherent in PSN.

2.3.6 Build request-response into the network

In IP, there is no notion of a “request” for data at a layer lower than the application. However, many user-level tasks (and therefore applications) make use of request-response semantics, e.g. web browsing or file sharing. In PSN situations, we often need to locate data of interest using dynamic and local connectivity rather than at a static infrastructure-based location. However, if requests and responding to requests were not part of the network, then we have situations as with the status quo where a webpage that I want may be on a computer next to me, but there is no way for my computer to ask for that webpage without relying on a particular peer-to-peer filesharing application being active.

To take another example, a mobile node might be at a location where it has no infrastructure connectivity, but it may wish to facilitate incoming data from other nodes, e.g. so it can receive email, or retrieve updates for the local web cache. By sending a request, it can cause other nearby nodes, which may for example have infrastructure connectivity, to act on its behalf, and eventually have the resulting messages propagated back towards it. This can lead to significant resource savings – if the requests include information on the current connectivity situation (e.g. sender’s location, nearby nodes, path the request took), then the responses can be directed more quickly and/or with a lower level of message replication (since successful delivery of each replica is more likely when using up-to-date network state information).

2.3.7 Exploit all data transfer methods

The aim of Huggle is to take advantage of all the communication opportunities offered by the PSN environment, including local connectivity with neighbouring nodes, and global connectivity using infrastructure. Human mobility patterns can be exploited by using forwarding algorithms which target nodes known to have mobility patterns which are likely to be useful, e.g. because they have seen a destination node recently [9], or because they share the same mobility pattern as a destination [7].

Between neighbouring nodes, there are potentially many interfaces that can be used, e.g. two neighbour nodes might have a Bluetooth, 802.11 ad-hoc mode, and infrared as potential connection opportunities. Huggle nodes must maintain a mapping of interfaces to nodes, since it is wasteful for two nodes to exchange data multiple times using different interfaces. Each connectivity method may have different properties in terms of bandwidth, latency, power consumption, etc, as well as having time-dependent channel characteristics such as congestion, so the choice of the correct connection method may be dynamic.

Infrastructure connectivity is not uniform either, with a particular infrastructure method having associated costs (which may be per-byte, per-time, or more complex schemes with varying rates, e.g. mobile phone contracts with a certain number of “free text messages” per month), as well as bandwidth, connection setup latency, per-message latency, etc. Some infrastructure methods are synchronous, e.g. when using direct TCP/IP between two Huggle nodes connected to the Internet. Some are asynchronous, e.g. the use of SMS text messages which are held until the recipient’s phone is on and has cell tower coverage. Huggle has to cope with both of these types.

2.3.8 Take advantage of brief connection opportunities

In the PSN scenario, connection opportunities can be fleeting, e.g. when walking or driving past another mobile user or an AP. It is therefore important for Haggle to be able to take full advantage of time-limited connection opportunities, by prioritising potentially exchanged data so that the most urgent data is sent first, and by using underlying protocols which make efficient use of bandwidth during short connection opportunities [5]. This also implies that neighbourhood discovery (neighbours meaning both mobile devices and APs in this case) is a key part of Haggle, since transfer opportunities must be detected in a timely fashion.

2.3.9 Empowered and informed resource management

Many of the principles above refer to resource management in some sense. Resource management is key to the success of Haggle since many of the proposals above have the potential to use up unlimited amounts of resources, e.g. data that is currently being held and forwarded for other nodes requires storage space, network bandwidth to send and receive, processing power to make and effect transfer decisions, and battery power to do all of the above.

This resource consumption might well be viewed a potential disadvantage of Haggle; for example, if a Haggle user's device were to run out of battery because it spends it all on forwarding others' data, that user will quickly disable Haggle. In fact, Haggle offers a unique opportunity to build resource management in to mobile devices in a scalable way, with minimal overhead for applications. Mobile devices often have plentiful resources. With battery life, many devices such as laptops have a "portable" rather than mobile usage model, and are plugged in when they are on. Storage capacity is growing at an exponential rate, with gigabytes already the reality, but typically users have to manually decide to copy data objects onto devices and personally manage the use of this large resource. Wireless networks have the great advantage of spatial reuse, but often only the space around APs is used, and away from APs there is much unused bandwidth being wasted, despite mobile devices moving through those spaces.

Storage

Storage resources are currently often used on a "first come first served" basis, and are only filled when applications specifically request it. This leads many devices to have most of the disk empty, so that they are "overprovisioned", and for devices which do run out of disk, the user often needs to manually find and remove low-importance data such as web caches. Haggle, since it keeps all user data, has the potential to manage storage space better, since some data is of clearly higher priority than others. For example, Haggle could flag each data item as "manual deletion only" (e.g. a document being edited), "delete if absolutely necessary" (e.g. a local cache of the user's old photo collection) or "deletion okay" (e.g. the web cache), and with priority levels, so that the data Haggle is holding in transit for a stranger is less valued than data held for someone who regularly communicates with the user of the device, i.e. a friend.

Networking

Networking resources must be managed for two reasons. Firstly, as discussed above, a particular connection opportunity may be of limited duration, and it is therefore important to prioritise the data sent using that scarce bandwidth so it is used to obtain the greatest benefit from the user's perspective. The second reason is that a given Hagggle node may have an almost unlimited set of networking tasks on its "to-do list", due to transfers in progress, as well as the need to use the network medium for neighbour discovery. To blindly execute the networking tasks in parallel or in FIFO order, as often done by network stacks now, will lead to low user-level goodput under high load (much of which may be speculative and replicated transmissions). Therefore, networking tasks should be carried out in an order determined by user-level priorities.

Battery

Battery resources are perhaps the most important to manage properly on a mobile device, as once spent they cannot be recovered without user intervention. Mismanagement can cause many problems for users, e.g. the inability to rendezvous with friends/family if your phone "dies." Users are therefore very protective of their battery life, and if Hagggle (and PSN technologies in general) are perceived to be battery-thirsty this might prove to be a key roadblock to deployment. It is less obvious is that, in some situations, users have *plenty* of battery resource. For example, while in a normal weekday routine, many people can easily charge their phone at night since there is a charger by their bedside. Since many phones normally require charging only once every few days, there is plenty of energy available if the users do not mind charging them every night in return for better application performance. Similarly, many laptops move from being plugged-in at one place to plugged-in at another, and are only on battery power for short periods of time.

For battery in particular, it is important to determine the "scarcity" of the resource — i.e. an estimate of how long it will be until the next convenient charging opportunity for the user. This can be achieved using *context-awareness* — by observing the patterns that the user exhibits at various times of day, the device's location, etc, a device can apply machine learning techniques to arrive at a prediction of how scarce the battery resource is. Thus, even if Joe User's battery is full, if Joe leaves his home city and heads to the airport, Joe's devices could infer that there may be no charging opportunity for some time, and therefore be conservative with battery consumption. Conversely, if Joe User's battery is only at 10%, but Joe will be home in 10 minutes and habitually plugs their device in on arrival at home, then perhaps there is plenty of battery to use for even low-priority application tasks.

Because Hagggle has the ability to centrally manage all networking and storage consumption of a mobile node, it is also the correct place for battery consumption to be managed and where control over its consumption can be exerted — e.g. a particular connection opportunity might be deliberately unused because Hagggle determines that the utility gained does not outweigh the battery cost.

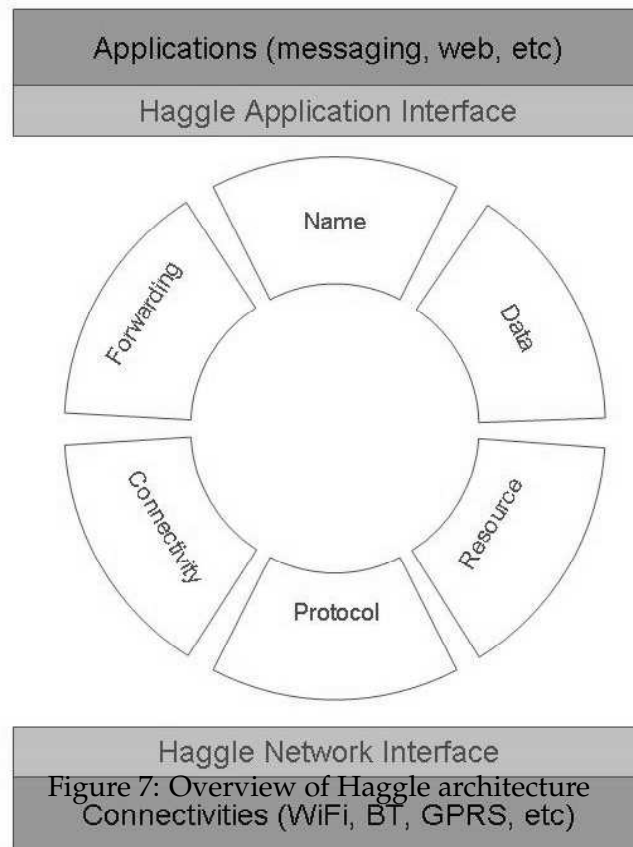


Figure 7: Overview of Haggle architecture
Connectivities (WiFi, BT, GPRS, etc)

2.3.10 Use and integrate with existing application infrastructure where possible

Haggle is not intended as an academic exercise in network architecture design, it is intended to be practical and useful. We must therefore pay close attention to existing deployments of applications and infrastructure for these applications, and integrate and reuse these. Haggle can gain three key advantages from doing so. First, Haggle is more easily incrementally deployed to users if they can interact with other users who do not yet have Haggle, via backwards compatibility. Second, users may wish to continue using the same, familiar application interfaces that they already make use of. Third, there is a vast infrastructure already deployed that will not change overnight, which Haggle must make use of in order to be competitive with the status quo.

3 The INFANT-Haggle Architecture

Haggle is the name given to our new network architecture, which applies the principles outlined above to overcome the problems with the status quo and effectively operate in the PSN

environment. Huggle is an unlayered architecture which internally comprises six managers; Forwarding, Data, Naming, Connectivity, Protocol and Resource, as shown in Figure 7 (we reproduced this figure here for better readability). As compared to the current network architecture in Figure 6, we immediately notice a number of high-level differences:

- User data is not isolated from the network, allowing it to be shared with other suitable nodes without an application being involved in each transfer.
- The application does not include network protocol functionality, making it easy for it to be agnostic as to the delivery method, and making the application code simpler.
- Huggle performs delivery using user-level names, allowing it to make use of all suitable protocols and network interfaces for delivery of a given data item.
- Huggle includes a resource management component which mediates between the other components using user-specified priorities to ensure efficient resource use.

We now discuss the functionality of the six managers in more detail and how they collaborate to achieve the goals listed previously.

3.1 Data Manager

Data Objects (DOs) are Huggle's format for user data, and as the name suggests they are an encapsulation for a data item meaningful to an application such as a photo, a music file, a webpage, a message, etc. DOs are not only the unit of data for applications, but are also the unit used for transfer between Huggle nodes, i.e. they are the messages in message switching. A DO is comprised of many attributes, where an attribute is a type-value pair. The type is always a text string, the value may also be a text string, but may also be a binary stream, e.g. for the mp3 file comprising the main content of a DO representing a song. The intent is for rich metadata about each object to be exposed as attributes — such data could be duplicated in the binary stream section if this is useful for the application.

A DO attribute can be used to store such information as:

- Descriptive information for user data, e.g. keywords for a picture.
- Document management information, e.g. the creation-date, creation-user, modification-date, etc.
- Security and permissions information, e.g. the list of local users with access permission, whether the item must be encrypted on leaving the node, and information to the encryption method and key

User data is often linked together to form compound data, e.g. a webpage which has embedded images. Huggle DOs can therefore be linked in a directional graph structure to reflect this relationship. In addition, applications and other internal Huggle components can claim DOs. When an application claims a DO, it is asserting ownership of that DO, allowing Huggle to

monitor resource usage by applications, and to not delete DOs which are of value to applications (and not marked with a low priority). Internal Haggle modules can also create and claim DOs, providing them with a general mechanism to store persistent data.

The Data Manager in INFANT-Haggle allows applications and other managers to insert Data Objects, to “claim” them and disclaim them, to insert links between them, and to search existing DOs using a “DOFilter”. DOs cannot be deleted per se, but if they are not claimed by anyone then they will be marked as lowest-priority and replaced by higher-priority DOs if storage space is an issue. DOs are immutable for applications, so they cannot modify a DO that another application is using. They are mutable for Haggle managers so that managers can edit DOs storing their own state.

An example DO is shown below.

Filename	DSC10027.jpg
Mime-Type	image/jpeg
Creation-date	1/1/2006 17:32
Created-by	James Scott
Security-group	Public
Keywords	Athens, Greece, seashore, sunset
Data	[binary jpeg data]

3.2 Name Manager

In Haggle, unlike in traditional IP networks, we do not assume that we can resolve a user-level name into an endpoint IP address before proceeding with data transfer. As such, we need to put high level naming information into in-transit data so that each intermediate node can make informed forwarding decisions.

Names are stored in INFANT-Haggle as Name Objects (NOs) which are special classes of DOs which have an attribute “Name”. We reuse the DO linking mechanism to allow names to be linked to other names, forming a directed graph of strings. NOs may be created and linked by applications (e.g. name to email address mappings from the contact list) by connectivities (e.g. using neighbourhood device discovery), by incoming data from other nodes (which may contain NOs to be integrated into the local Name Manager). The flexible nature of DOs also means that different naming schemes can be constructed easily, enabling the use of Haggle for new applications and protocols in a way that highly-specified lookup tables (e.g. DNS MX records) do not.

An example NO graph might be “James Scott” NO which links to two child NOs, namely “james.w.scott@intel.com” and “00:0E:F2:98:2A:B5”. This represents the person “James Scott” who is reachable by both an email address and a MAC address for their device. Names are useful for delivery because of the neighbour discovery process discussed later, which results in certain names being regarded as “nearby” (i.e. deliverable-to by a protocol) at any time (e.g. the email address is “nearby” if the node currently has Internet connectivity).

The Name Manager’s job is to allow applications to insert, retrieve, and link between Name Objects, and to merge in naming information obtained from other nodes either proactively or through the inspection of objects that are “passing through”.

3.3 Forwarding Manager

Forwarding in INFANT-Haggle requires applications to specify a set of Name Objects for the destination and a set of Data Objects for the data to be sent. In INFANT-Haggle we do not yet support “data solicitation” i.e. “pull” operations to retrieve data from other nodes using a keyword filter, though this is planned for a future release.

When an application or other Haggle manager requests a forwarding operation to commence, the Forwarding Manager creates a Forwarding Object (FO) which is a type of DO. These link to the DOs being transferred as well as the NOs representing the destinations (provided in the specification of the forwarding operation). The types of data that can be stored in an FO include:

- details of the source node (e.g. NOs belonging to the source node).
- a list of forwarding hints, informing intermediate nodes of dynamic information (e.g. X-was-seen-at-time-T-by-Y).
- a list of nodes which it has passed through.
- timeout/flooding avoidance parameters — e.g. max duplication count, max hop count, deadline.
- a priority level specified by the source node.
- security information, e.g. an authentication signature or encryption details for the claimed DOs.
- any other data relevant for forwarding tasks — since the DO format allows a variable number of attributes, a new forwarding algorithm can easily send other data.

Novel forwarding algorithms are a key research area for Haggle, and the DO structure provides flexible support for forwarding algorithms. This is achieved by storing forwarding state in DOs as well as application data described above. Unlike with packet headers in traditional protocols, DOs are a flexible data format that can easily be modified to add or remove fields.

Forwarding algorithms in Haggle estimate the “benefit” of performing a transfer of a given FO to a given “nearby name” (provided by the Protocol Manager — see below). While some transfers are obviously beneficial, e.g. the transfer of a DO to an email address which it lists as a destination, other transfers are less obvious, e.g. the transfer of the same DO to a node which is not the final recipient but might be willing to help in the transfer process.

Within INFANT-Haggle we currently only implement a DirectForwardingAlgorithm, i.e. only sending an FO to a nearby Name if that Name is listed as a destination in the FO itself. In future we hope to integrate forwarding algorithms that are the result of the partners’ research into opportunistic and pocket switched networking scenarios, e.g. leveraging human communities in order to provide a forwarding path.

3.4 Connectivity Manager

As previously mentioned, Haggle must perform neighbour discovery in order to take advantage of connection opportunities. The result of neighbour discovery is a set of “neighbours”. One example might be 802.11’s discovery process, which would result in a set of 802.11 MAC addresses as neighbours. Furthermore, each neighbour is associated with a boolean flag “isInternet” which states whether there is Internet connectivity available through this neighbour. For 802.11 access points, this flag might be set, while for 802.11 devices visible through ad hoc mode, this would be false.

The Connectivity Manager’s job is simply to be a wrapper for a set of Connectivities, to instantiate these Connectivities when Haggle starts up, and to allow other Managers to access the list of running Connectivities.

Each Connectivity is responsible for a particular physical network interface. It must perform neighbour discovery on this interface (via the Resource Manager’s task interface described later), it must provide data connections to these neighbours when demanded by Protocols, and it must determine the Cost of using an interface for particular Tasks so that the Resource Manager can perform Task selection (see description of the Resource Manager later).

3.5 Protocol Manager

In Haggle, formerly application-layer protocols such as SMTP/POP are now executed inside the Protocol Manager instead of in the apps themselves. This allows the choice of the appropriate protocol depending on the connectivity situation - e.g. a message can only be sent via SMTP if Internet access is present. In INFANT-Haggle the Protocol Manager simply provides a wrapper for a set of Protocols, and must instantiate these Protocols when Haggle starts up, and provide access to the Protocols. Each Protocol registers for neighbourhood information callbacks from the Connectivities they wish to use, and when neighbours are discovered, Protocols must determine which Names they can send to via a Neighbour (perhaps inserting a new Name if the Neighbour has not been seen before). For example, if a device X nearby is discovered via ad hoc mode, then the Name X is conceptually marked “nearby” by the Peer-To-Peer Protocol and the Forwarding Manager knows that that Name is reachable. Another example: when Internet access is detected by a Connectivity, the EmailProtocol would mark all email addresses as “nearby” since it knows a method of sending data to them.

Protocols are also responsible for sending and receiving data. Sending is triggered by the Forwarding Manager (via Tasks issued to the Resource Manager), while receiving is up to the Protocol itself (and again involves issuing Tasks to the Resource Manager to authorise reception). Both proactive (e.g. email checking on a server) or reactive (e.g. incoming P2P connection) reception is possible.

3.6 Resource Manager

All use of resources in Haggle is controlled by the Resource Manager. This operates by performing a cost/benefit analysis on “Tasks” that other modules specify. Tasks include pointers to callbacks for obtaining the current benefit and the current cost of that task. The benefit is

a scalar value representing the relative importance of a task as a percentage, while the Cost is expressed in INFANT-Haggle in terms of the time spent on a network - in future this will be expanded to include monetary and energy costs too.

By using this abstract description of Tasks, we allow operations by different Managers and different modules within the Managers (e.g. a Protocol) to be compared on an equal footing. At any one time, a Haggle node might be able to do neighbour discovery, check email, send some data to a peer, send some forwarding signalling information to a peer, broadcast some naming information, etc. Note that the benefits and costs are both dynamically evaluated every time a Task is considered, since they may change — e.g. the benefit of checking email on a server drops if email was checked recently, and the cost of using an AP to send something in 802.11 drops if the node is currently associated with the AP.

3.7 Interacting with applications

The Haggle architecture provides a new abstraction layer for mobile applications, at a much higher level than the “socket” abstraction that is currently used. The key properties of the Haggle architecture from the application’s point of view are:

- Haggle supersedes the file system on mobile devices, providing a persistent storage abstraction for Application Data Units (DOs) which allows applications to specify a rich set of metadata governing how that data is used.
- Haggle provides applications the ability to specify networking tasks based on the contents of DOs, e.g. asking Haggle to retrieve DOs that are photos of a particular place and time, or webpages with certain keywords.
- Haggle abstracts the networking facilities of the mobile device so that the application does not need to implement particular transfer protocols which are infrastructure-centric, and can instead transparently make use of neighbourhood, infrastructure, and mobility-based data transfer opportunities.
- Haggle provides a way of naming and addressing entities which are not single network nodes (as with IP) but are high-level concepts such as people, places, services, or information, using trees of Name Objects.
- Haggle allows applications to specify priorities for tasks, so that the limited resources of mobile devices can be spent for maximal user benefit, and spare resource can be used for secondary tasks (e.g. web prefetching) while not jeopardizing the primary tasks (e.g. urgent messaging).

4 Integration and Testbed

4.1 Implementation Environment

We chose to conduct the implementation on the open source website Sourceforge.net immediately from day one. This provides useful tools for development (email lists, web space, ability to do file releases, cvs web access, etc) and also provides the project with immediate visibility to the outside world. It also fits perfectly with the spirit of the Haggle project, which is to produce open source software. The software is currently licensed under the GNU GPL, although we can also use other licenses in future if this becomes useful.

The Java programming language was chosen for cross platform support and because there are a large number of open source tools in Java that can be reused. The Eclipse programming environment was chosen due to its easy availability and good support for Java.

The first step in integration is to provide a modular architecture so that individual components can be integrated and modified without affecting other components. This architecture is represented in the existing codebase using Java interfaces. The second step in integration is to provide "reference" implementations of these components, and has also been achieved. We implemented the Data Manager using an SQL database, 802.11 Connectivity in both AP and ad hoc modes, both an EmailProtocol and a PeerToPeerProtocol, and also implementations of the other managers.

The third step, also achieved, is to construct a demo application that illustrates Haggle functionality in the PSN environment - this application is Messaging and is discussed below.

4.2 Haggle Testbed

Our first step will be to extend the APE testbed to include other portable devices such as the Linux-powered Nokia 770 and then port the Haggle architecture implementation on it. A Ph.D. student will begin working on this starting August 2006. For better manageability and coherent testing, we hope to create a testbed so that all devices (laptops, Nokia 770, etc.) run a common (Linux) environment, though they should be able to interoperate with other platforms if needed.

5 INFANT-Haggle Application: Messaging

Haggle enables a new family of applications with a high degree of spatial or logical locality. Haggle takes the aforementioned communities as both inspiring models and connectivity hypothesis to lay down new forms of networking, and provide explicit support for community formation and management within its context. The first demonstration and implementation application for Haggle was chosen as asynchronous messaging. This application type includes email and SMS, possibly the most popular applications for existing laptops and mobile phones respectively. It is well suited to Haggle since users already comprehend that their message may take some time to deliver, hence they are already primed to consider the message latency that may be inherent in autonomic networking scenarios such as PSN.

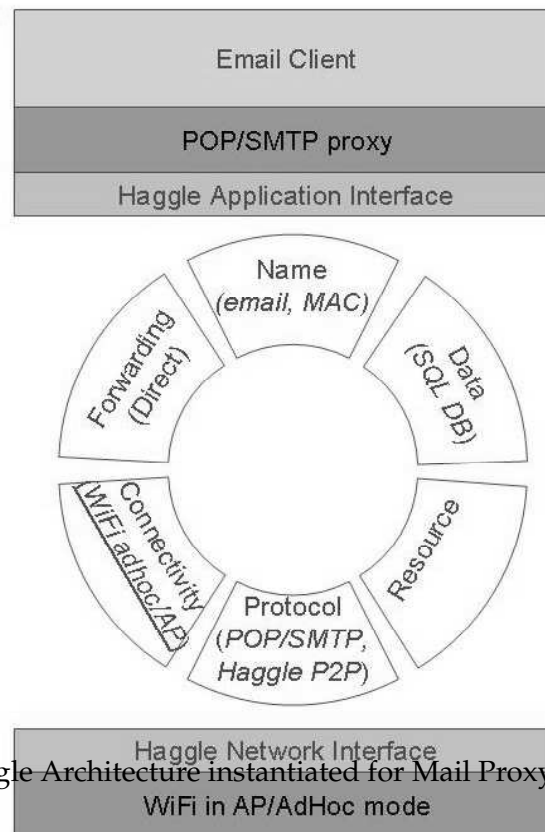


Figure 8: Haggie Architecture instantiated for Mail Proxy application

While applications should ideally use the Haggie Application Interface directly, to achieve the benefits stated previously, for the INFANT-Haggie we chose to reuse existing messaging applications such as Mozilla Thunderbird or Microsoft Outlook, by using a *proxy* between these applications and Haggie. This has multiple benefits:

- It demonstrates legacy compatibility for Haggie since existing applications can be used.
- It allows us to see how users react to Haggie's opportunistic networking functionality using a single, familiar application rather than asking users to compare using a different application under Haggie to under traditional networks.
- It eases deployment since users do not have to change their applications to obtain the benefits of Haggie.
- It allows Haggie to reuse the user interface design and GUI work that have gone into these applications, saving us additional engineering effort.

The proxy is in the form of a Haggie application named "MailProxy" which provides SMTP and POP server functionality which client applications such as Thunderbird can connect to

over “localhost”. Outgoing and incoming emails are stored in Haggle as DOs. The login details provided to the Haggle POP server are passed to the EmailProtocol for use in receiving email from Internet POP servers. The sender email addresses provided to the SMTP server are passed to the Name Manager as “self identities”, i.e. they define Names which belong to that node. This proxy application has been implemented and is available with the rest of the code at <http://www.sourceforge.net/projects/haggle/>. An illustration of the proxy can be found in Figure 8.

6 Annex I - Acronyms

AP	Access Point
CM	Communication Model
CVS	Concurrent Versions System
DO	Data Object
FO	Forwarding Object
GUI	Graphic User Interface
HCM	Human Communication Model
NC	Network Coding
NO	Name Object
PSN	Pocket Switched Network

7 Annex II - Code

The code of INFANT-Haggle node is fully available at <http://www.sourceforge.net/projects/haggle/>.

References

- [1] Giuseppe Anastasi, Marco Conti, and Andrea Passarella. Power management in mobile and pervasive computing systems. In Azzedine Boukerche, editor, *Handbook of Algorithms for Wireless Networking and Mobile Computing*, chapter 24, pages 535–576. Chapman & Hall/CRC, 2006.
- [2] R. M. Berko, A. D. Wolvin, and D. R. Wolvin. *Communicating, A social and Career Focus*. Patricia A. Coryell, 2004.
- [3] David D. Clark. The design philosophy of the DARPA internet protocols. In *Proceedings of ACM SIGCOMM (Computer Communications Review Vol 18, No 4)*, 1988.
- [4] Joseph A. DeVito. *Essential fo Human Communication*. 2005.
- [5] Richard Gass, James Scott, and Christophe Diot. Measurements of in-motion 802.11 networking. In *IEEE WMCSA 2006 (to appear)*, 2006.
- [6] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. Pocket Switched Networks and human mobility in conference environments. In *Proceedings of the SIGCOMM 2005 Workshop on Delay-Tolerant Networking (W-DTN05)*. ACM.
- [7] J. Leguay, T. Friedman, and V. Conan. Evaluating mobility pattern space routing for DTNs. In *Proc. INFOCOM*, 2006.
- [8] Christoph Lindemann and Oliver P. Waldhorst. Effective dissemination of presence information in highly partitioned mobile ad hoc networks. In *Third Annual IEEE Communications Society Conference on Sensor, Mesh and Ad hoc Communications and Networks (SECON 2006)*, Reston (VA), USA, September 25-28 2006.
- [9] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. In *Proceedings of The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, August 2004.
- [10] Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–26, January - March 2005.
- [11] Luciana Pelusi, Andrea Passarella, and Marco Conti. Beyond MANETs: dissertation on opportunistic networking. Technical report, IIT-CNR, 2006.
- [12] Luciana Pelusi, Andrea Passarella, and Marco Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *submitted to an International Journal*, 2006.
- [13] Roy Want, Keith I. Farkas, and Chandra Narayanaswami. Guest Editors' Introduction: Energy Harvesting and Conservation. *IEEE Pervasive Computing*, 4(1), January - March 2005.
- [14] Liangzhong Yin and Guohong Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(1):77–89, January 2006.