HAGGLE: Opportunistic Communication in the Presence of Intermittent Connectivity

Eben Upton, Marc Liberatore, James Scott, Brian Levine, Jon Crowcroft, Christophe Diot
Intel Research Cambridge, UK

August 26, 2004

Abstract

We describe HAGGLE, a new networking architecture designed to enable communication in the presence of intermittent connectivity. By intermittent connectivity, we mean any type of network connectivity, including (but not limited to Blue Tooth, 802.11, Ethernet) whether it is local or connected to the Internet. We justify the necessity for HAGGLE with some simple usage scenarios where service cannot be provided with today's Internet technologies (or can be provided with unnecessary complexity). We then present a system that uses best-effort, context aware message forwarding between ubiquitous mobile devices to provide service in these scenarios. We describe the various components of HAGGLE architecture among which privacy, authentication, trust, and incentive to cooperate play a special role. We discuss the feasibility of HAGGLE. We identify implementation challenges and describe a prototype implementation.

1 Introduction

The dominant communication environment is the Internet (not considering networks dedicated to specific applications such as the POTS). Most Internet services rely on the end-to-end principle, i.e. a sender and a receiver must have agents simultaneously connected to the Internet for a successful transaction to occur. Applications and protocols are designed in this fashion, including core services such as the web. When in fact very few applications require end-to-end connectivity, namely interactive services (Instant Messaging, VOIP, gaming, video streaming, etc.). However, in a world of cell phones, PDAs and laptops, devices are often disconnected, making Internet services inconvenient to use. With mobile devices increasingly equipped with various wireless networking technologies[1], more applications are actually being used whilst mobile (e.g. in airports, on foot, or in the train). The direct consequence of mobility is that such devices may often (but temporarily) be out of range of the Internet infrastructure, and thereby unable to access Internet services. However, these devices may often be in range of other networked devices, or pass by an area where some type of connectivity is available (e.g. WiFi Hotspots) and may be able to perform useful networking transactions involving local exchanges of data.

In fact, [**?**] showed that full connectivity is rare for mobile users with limited radio range and limited density of nodes. In addition, NAT has been challenging full connectivity for quite a while now.

In order to overcome the limitations of communication services being provided between fixed end points connected to the Internet, we propose HAGGLE, a communication architecture designed to provide *opportunistic communication services* to mobile end points intermittently in reach of other devices or networks. We believe HAGGLE can provide a range of networking services including point-to-point communication, content sharing, and queries, despite the lack of systematic connectivity to the Internet. This is accomplished using a distributed information space model, where data objects can be created locally, and propagated through both local and Internet connectivity as may be available, subject to some forwarding policies. HAGGLE builds on the fact networking devices are willing to cooperate to convey data through their destination despite the absence of Internet connectivity (or in order to reach an Internet gateway). As such, it is similar to the famous small world model [15].

[1] such as 802.11a/b/g and BlueTooth today, and others under development, including new 802.11 standards and ultra wideband

A fundamental assumption we make is that storage (and to a lesser extent power) will soon not be a significant issue in mobile devices, and that this will reduce the barrier to cooperation.

Critics will claim that HAGGLE is yet another ad hoc network architecture. We claim that most other ad hoc networks are based on the flawed assumption of end-to-end connectivity, and that HAGGLE is among the first architectures that do not make this assumption. Rather, in HAGGLE, we assume (i) devices are usually disconnected, (ii) devices cooperate, (iii) there is no centralized infrastructure to authenticate a device or to match a name, and (iv) devices can usefully answer one another's queries, independent of the Internet.

The paper is organized as follows. We first illustrate the HAGGLE domain of application with a number of motivating scenarios. Then we sketch a high-level description of the constraints a networking system will have to function in a partially connected world. Next we look at related work. We then describe HAGGLE and its components, including the use of communities, the need for trust and security features, and the issues involved in developing useful applications for deployment to a wide user base. We then describe an implementation of HAGGLE currently being developed,and future plans for this research.

2 HAGGLE Overview

In order to motivate HAGGLE and its architecture, we present five scenarios where the Internet and its end-to-end architecture fail (or make it difficult) to deliver a given communication service. We then briefly describe some of the constraints within which HAGGLE will function.

2.1 Scenarios

- Alice is attending a conference without wireless access. She meets a colleague, Bob, and after some conversation she decides she wants to forward Bob a document which she received by email. She initially tries to forward the email, before realizing that this will not work with her client in disconnected mode. It takes 30 minutes for her and Bob to discover that both their devices have Bluetooth, configure their software, and send the document using this method (after trying and failing with IR, as usual). HAGGLE would provide here a solution to transparently forward Alice's email to Bob in a secure way. Note that in case Alice and Bob would not have the same connectivity interface, HAGGLE would transfer the email asyncronously, seamlessly, and securely via one or more third party devices.

- Charlie is finally nearing the front of a traffic jam. After half an hour of wondering what caused him to miss dinner, he can see the oil split further up the road. More and more people are arriving at the back of the jam, but Charlie, despite the presence of a PDA, a cell phone and a laptop in his car, has no way of letting them know a) what is causing the jam, b) how long the queue is, and c) to take an alternate route. However, local connectivity could be used to haggle the information among people caught in the traffic jam, or to contact the police. HAGGLE supports here local broadcast of information.

- Donna is atop the Eiffel tower, and takes an amazing photo with her camera-phone. Another tourist, Ed, is there too. His phone has no camera. He wishes it did, or that he could haggle Donna's photo to take home. Note the negotiation phase needed in this transaction (that is present in most scenarios), justifying the name of HAGGLE for the proposed architecture.

- Fred arrives at the airport. He wants to send the messages he has written during the previous flight. He does not have time to stop by the next phone to dial-up and send his email messages. However, he walks by George who is currently connected on the local WiFi Hotspot. George could easily forward Fred's message with a minimal impact on his connection, and after some verification on Fred or letting Fred open a VPN through George's computer.

- Jim is attending a conference. A person addresses him very familiarly. Jim does not want to offend this person. He takes a picture with his cell phones and haggles the picture around asking "do you know who that guy is?" This scenario illustrates content-based service request rather than directed to a specific destination or locally broadcasted.

For all scenarios, it is clear that some flavor of wireless networking could solve the various users' needs, but that such solutions are difficult with current end-to-end technology. It is possible to cater to each situation with a jury-rigged solution, (e.g. a manually initiated infrared transfer from Donna to Ed), but this creates a headache for users who have to be aware of each solution and the appropriate

situations to use it. The goal of HAGGLE is to provide a unified framework to allow applications to easily use local networking as well as end-to-end networking, and thus much more easily support the activities described above.

2.2 Constraints on HAGGLE

HAGGLE takes an orthogonal approach to the Internet and gives a new significance to the phrase ”best effort”. The HAGGLE communication architecture will have the following properties:

- HAGGLE devices must take advantage of any connectivity (opportunistically) and of the presence of any other device in its neighborhood.
- Because of mobility and no guarantee of end-to-end connectivity, building routes is not possible. Therefore HAGGLE must be a one-way communication architecture where each node forwards messages based on local context information (e.g. current neighborhood, information about past transactions, information provided by remote node).
- There is strictly no guarantee that a service is delivered. In the Internet, many services are built atop a generally reliable service even though they may provide no strong guarantees. For example, users regard email as reliable, despite the potential for non-delivery. HAGGLE is instead an unreliable best-effort service, which is the best we can achieve in intermittently connected environments. Acknowledged services can be designed on the top of HAGGLE, but HAGGLE is in essence a one-way communication architecture.
- For the preceding reasons, a HAGGLE node can not rely on a global naming or authentication services. It needs to forge its own opinion using data available locally (local state information and neighborhood).

We believe that the best way to work within these constraints is a radical integration of a general application communication framework with the network protocol. There will be no layering above the link layer, as the link layer is all we have access to. Instead, messages contain all the information necessary to send them to other devices. In other words, *the application is the network*.

In order to avoid confusion with other communication architectures, a HAGGLE message is called a ”service request”, and ”haggle” becomes a verb. Each application will use a unique service request format and transmission service defined by HAGGLE. The structure of service request and the details of how they are haggled is described next.

3 Related Work

Several existing systems are related in some fashion to the HAGGLE. We briefly describe those systems here, and point out where their assumptions differ from those of HAGGLE.

3.1 Mobile Ad-hoc Networks

Mobile Ad-hoc Networks (MANETs) [19] are an attempt to utilize the bandwidth that is available to wireless devices. Research into MANETs has been progressing for nearly a decade [11, 17] with no significant end-user deployment in sight. It is difficult if not impossible for end-users to utilize that which only exists in simulation. Of the older and presumably more mature MANET protocols, AODV [18] is one of the few that has a reasonable set of real-world implementations [1, 20]. Despite available code, there has not to our knowledge been any large-scale adoption of this technology in the wild. **FIXME:** The largest experiment we are aware of (120 intermittent participants) took place during IETF XX meeting in March 2004[2] and mostly helped confirm the limitations of AODV.

We believe that there are several reasons for this lack of real-world deployment. The design of most MANET protocols is informed heavily by the end-to-end nature of the Internet. In general, MANET protocols attempt to build routes between nodes by either maintaining routing tables, as in DSDV [17], or by forming routes on an as-needed basis, as in AODV [18]. Though often left unstated, a presumed goal of this route formation is to reach a bridge to the wired Internet and enable end-to-end service for a mobile node.

In addition, connectivity in ad hoc requires a minimum node density (c.f. Kumar/Xie and others, as well as Penrose) in order to build the MANET. HAGGLE shortcuts this need as it is designed to function in intermittently connected conditions.

We believe that MANET is a flawed approach. First, MANET was designed to be Internet compatible. As a consequence, they could not take into consideration the intermittently connected nature of ad hoc networks. The performance of TCP/IP over multi-hop wireless links is

[2] www.cs.ucsb.edu/xxxx

generally poor [10], and in general, end-to-end communications over multi-hop wireless networks is difficult [14]. Further, shorter range wireless technologies and intermittent connectivity spoil attempts to form and maintain routes over time, or to keep any resilient state about network neighborhood. Reaching access points and maintaining connectivity is an increasingly unlikely proposition as paths grow longer.

These points indicate that a move away from traditional end-to-end services will be appropriate for MANETs of the future. While legacy systems like the Internet will certainly be exploited when available, they will not be the focus of such MANETs.

3.2 Delay Tolerant Networking

The Delay Tolerant Networking Research Group (DTNRG) [8, 9] is examining routing self-contained messages ("bundles") through networks with long delays, high error links, and frequently-disconnected, pre-scheduled, or opportunistic link availability. As in HAGGLE, DTN proposes messages that contain information about service requirements, though there is little or no notion of using application-level information to assist in forwarding decisions.

HAGGLE and DTN have a similar goal: message delivery in a intermittently connected networks. DTN is still conservative in its current approach, as it aims to provide end-to-end service via a message-based overlay and reliable transfer of delivery responsibility. HAGGLE clearly is a more radical approach, breaking the assumption of end-to-end connectivity and merging application and communication into a single architecture. Another major difference is that HAGGLE's focus is not delay but intermittent connectivity. Therefore, HAGGLE could be qualified as a CTN technology (Connectivity Tolerant) instead of a DTN one. However, DTN and HAGGLE share many issues and HAGGLE could learn a lot from DTN past experience.

3.3 7DS

7DS [16] is a system designed to provide MANET-like functionality in a partially disconnected world. 7DS permits mobile nodes to access content (identified by URLs) through a combination of pre-fetching and a sharing of cached data among nodes. In this fashion, some object requests can be satisfied when a node does not have a path to the true source of a URL. 7DS conserves power by having nodes refrain from answering requests from objects when power supplies are low; there is no explicit incentive to cooperate. Unlike HAGGLE, 7DS is not designed to allow any networked device to take advantage of any local connectivity opportunistically to provide a large range of DTN-like services. Instead, it uses 802.11-based multicast to find nearby caches of popular content.

3.4 Freenet

Freenet [6] is a system that was developed to allow anonymous file sharing among participants on the Internet. Content is named in a distributed fashion; this allows for scalability an anonymity at the expense of a search capability. Nodes send requests for information out into the Freenet network. These requests are propagated until an answer can be found, at which point it is propagated back. With some probability, nodes along the return path will cache the content, thus making the discovery of the true content publisher difficult. The Freenet authors are also currently working on smarter message forwarding algorithms that take into account not only node contents, but response time, as well. Distributed naming and lack of end-to-end connectivity are strong components of Freenet, as well.

3.5 Top-Down Approaches

[[[CHD: Please rewrite with proper names and reference. this is a place holder.]]] In addition to the above architecture project (that could be qualified of bottom-up), there are numerous Top-Down approaches of interest. These approaches solve all subsets of the problems solved by HAGGLE and could be added to the list of scenarios. In Zebranet [12] zebras are instrumented with collars that collect data about zebra's location, shape, etc. These collars communicate in order to opportunistically merge the collected information in order to bring it back to a base station where data can then be analyzed. Lapnet [**?**] is a similar initiative. Nomadic populations in the north of Sweden carry WiFi base station on their skidoo. Data are opportunistically transferred to the Internet when the skiddo goes in reach of a collection station installed around Lapland. Cyberpostman [**?**] is an experience in vietnam to bring connectivity to remote villages through the postman. Local computers in the villages now that they can transfer their Internet data while the postman is in their village. The postman will in turn transfer the collected data to the Internet once back at the postoffice. Last, estadium is a Purdue university project. An American football stadium has been engineered with 802.11 access points. Spectators can use their PDAs to watch videos and exchange emails during games.

Note though that most of these project (except ze-

branet) are extensions of the Internet to support intermittent connection. They do not extend the range of applications and services currently available on the Internet. In addition, they all rely on the deployment of a specific infrastructure. HAGGLE does not require such deployment and can function in a purely opportunistic mode (i.e., not only the postman could collect the data from the villages, but any visitor).

4 The HAGGLE Architecture

In order to allow users to opportunistically take advantage of local connectivity, it is not sufficient to provide standard socket-based networking support. It is necessary to depart from the nature of currently used networking APIs, which are node- and connection-centric rather than data-centric.

In this section, we describe several of the key conceptual and architectural components of HAGGLE, highlighting the inter-relationships that we believe show the necessity of unifying the application and the network.

4.1 Fundamentals

4.1.1 Information Space and Message Format

HAGGLE is based on a globally distributed information space comprised of a tuple space with a set of attribute-value pairs describing the set of information that a HAGGLE node will use to convey a given service request. The local information space on each device can be added to and read from by local HAGGLE applications with appropriate permissions. Therefore, the local information space continuously evolves with the neighborhood of the node, with time, with connectivity, with a node user's own information, etc.

This semi-structured information space allows us to leverage existing techniques for information retrieval when the user or other nodes make service requests.[3]

We refer to objects exchanged by HAGGLE capable nodes as ***service requests***. Those queries are constructed of attribute-value pairs, as described above. Attributes may include keywords describing the content, the data's file format, the intended and/or allowed recipients, timestamps such as time-of-creation and time-to-live, data to help or control message propagation decisions, information on the value of an object (ranging from manual-deletion-only to first-to-go), etc.

[3] Ad hoc google!

Some examples of possible HAGGLE messages are shown next.

```
*******************************************

service-request-type: private message
originator: NAME James Scott
sender-authentication: [cryptographic signature]
send-to: NAME Eben Upton
data-type: text, encrypted
data: [encrypted set of tuples with message]

service-request-type: relay
originator: NAME James Scott
sender-authentication: [cryptographic signature]
send-to: SMTP eben.upton@intel.com; eupton@yahoo.co
send-to: SMS +441223763456
send-to: NAME Eben Upton
data-type: text, clear
data: I will be late for dinner. I am stucked in a

service-request-type: search
originator: NAME James Scott
sender-authentication: [cryptographic signature]
protocol: Google
data-type: query
data: search?q=Air+France&sourceid=mozilla-search&s
ie=utf-8&oe=utf-8

service-request-type: locate
originator: NAME James Scott
originator: SMTP james.scott@intel.com
sender-authentication: not provided
find: NAME Jon Crowcroft
find: JPEG j.crowcroft.jpg

************************************************
```

In HAGGLE, there is no minimal or maximal set of attributes. Attributes are interpreted by each HAGGLE node based upon the node's functionality and willingness to cooperate. For example, if an attribute says `send-to NAME John Smith`, the next hop will be defined solely based on local node information about John Smith. To increase probability for a query to reach its destination, its initiator can add multiple destination identifications such as `send-to NAME John Smith , SMTP jsmith@world.org , IMAGE jsmith.jpeg`. This example demonstrates a name, an email address, and

a picture all to be used in each intermediate HAGGLE node's forwarding decisions, if possible. Note that a HAGGLE message does not necessarily have a destination. For example, a message may only contain a source id and a question for a search engine.

We identify the following open problems:

Total As much of the node's state as is practical is stored in the shared space, facilitating easy exchange of this information.

Tuple Format Largely string based, allows us to store information in a structured format, perform explicit database-style queries, and use existing IR techniques to answer Google-style queries.

Subset Messages The message format is just a subset of the information space format.

Last Seen Track the last seen time of nodes (P{see soon|seen recently} $>$ P{see soon}?).

4.1.2 Opportunistic Service Request Propagation

HAGGLE service requests are propagated from node to node based on the capability of a node to match a service request's attributes to its local information and the likelihood of the node encountering another node with an answer in the future. Upon receipt of a query, a node must decide among the options:

- Display to the local user of the node,
- forward & discard
- store & forward
- ignore (discard)
- provide service

Flooding is **NOT** the default propagation paradigm in HAGGLE. Instead, HAGGLE nodes try first to choose the next hop using their local interpretation of a service request and their knowledge of nodes in their neighborhood. Other portions of the HAGGLE architecture provide information to the forwarding engine so as to maximize the chance of delivery. There are 3 types of forwarding in haggle

1. Bootstrap is just used to discover a node's neighborhood. This is used for crawling and indexing in the google type part of haggle as well as for bootstrapping data for the other two types of forwarding. Bootstrap can rely on flooding or broadcasting if the communication technology permits it.

2. Gradient based forwarding where gradient is tora like but based on application/user choice of metric to apply (e.g. familiarity, trust or signal strength or whatever

3. "luckless" forwarding where a response is requested and so some type of redundnacny/persistence is used to forward along multiple routes and the response may arrive from several places (like bittorrent, even, with different pieces coming from different sources.

Note that we are not proposing that the Internet is no longer useful, nor are we supporting the other extreme that reaching the Internet is the goal of HAGGLE messages. From HAGGLE standpoint, The Internet is considered as another type of connectivity (such as POTS, hotspot, etc.) that can be used to opportunistically convey a service request where available and if information provided in the service request make it possible. Consider a device which is carrying a HAGGLE service request, and sees an Internet access point; it can decide to send the message as an email using this infrastructure, given the appropriate information is available. The same device can also keep a copy of the message and deliver it through another path to its destination. This situation illustrates that, in today's world, both infrastructure-based and opportunistic messaging are required in order to optimally deliver content.

We identify the following open problems:

Push or Pull? How best to share queries when connection time is limited?

Supernodes Do explicit supernodes (e.g., hardwired PCs) improve forwarding?

Mobile Code Constrained mobile code may allow for more efficient message propagation or queries.

4.1.3 Community-based Networking

We expect HAGGLE to enable a new family of applications with a high degree of spatial or logical locality. We refer to these areas of networking as communities, and provide explicit support for community formation and management within HAGGLE. We believe that this notion of community will make propagation of information in HAGGLE easier to achieve and control. Service requests may have more success if forwarded preferentially among through a community. Communities can help improving the security of transactions. A node might give preferential treatment to a service request related to a community for which this node has state information. Examples of

such communities are participants to a symposium, fans of Elvis Presley, London subway users, etc.

Communities are formed in a distributed fashion. The impetus for community formation may be implicit or explicit. An example of the former would be a common geographical location or a common interest, while communities may form in the latter manner around a given event, such as a conference. To form communities based upon space-time proximity, a TTL field in messages may be appropriate. Other communities with more tightly controlled access may allow open membership, vote to decide upon membership, or appoint a trusted authority to control the membership of the community. Communities might provide some information that can later be used by a node to make a decision (i.e. in a conference, the list of registered participants can be provided prior to the events). Community is also a convenient paradigm to secure HAGGLE: nodes can create shared keys to allow for private communications, and utilize known techniques for key revocation within the community.

We identify the following open problems:

Time-Space Proximity Nodes that repeatedly see one another over a short enough time scale form a community.

Interests Through IR techniques, extract interests based on user's content.

Specialty Extraction E.g., having an `@intel.com` email address is a form of community membership.

Managed Assigned through a trusted third party.

4.2 Security

Security is a major issue in HAGGLE and several important security services must be provided to users. In this section, we discuss issues related to secure distributed naming, authentication, trust, reputation systems, and incentive to cooperate.

4.2.1 Identity and Trust

A user can have one or more identities that they will use within HAGGLE. Some of these identities will be specific to a community. On the other hand, they are not device-specific and a HAGGLE user can use the same identity on various HAGGLE capable devices. An identity can be (non exhaustive list) an email address, a URL, a names, a picture, or any combination of the above.

Identities will be tied to public/private keypairs generated by the user, and may be shared or moved between HAGGLE nodes. In traditional public key cryptosystems, the bootstrapping of trust[4] is a difficult problem, solved by either a trusted third party [4] or a distributed web of trust, such as that in PGP [2]. HAGGLE is particularly well suited to easy deployment of a web of trust, as users will be carrying small devices and can quickly and conveniently bootstrap trust between one another [13]. In the case of more managed communities (see section 4.1.3), a trusted third party can serve as an authenticator of identity. This notion of identity will serve as a building block for more advanced security services, as other users will attach notions of trust, reputation, content, and capability to an identity.

We identify the following open problems:

Multiple UIDs email addresses, names, pictures, etc. Tie them all to a PGP-style public key.

Web of Trust Allow secure verification of keys between users and nodes.

4.2.2 Reputation System and Incentive to Cooperate

HAGGLE will provide a reputation system to discourage malicious behaviors. Reputation systems have a different (thus complementary) purpose than trust control. Trust control is about getting the guarantee a user really is who he claims to be. While reputation is about quantifying how good a HAGGLE citizen a node or user is.

There is a negotiation aspect embedded in HAGGLE (hence the name of the communication architecture). The negotiation phase can take various aspects. At bootstrap time, nodes can negotiate in what condition they accept to forward, or provide service. Such negotiation can also take place when a node receives a service request, depending on the content of the service request. Negotiation might not be explicit. HAGGLE must be able to adjust reputations not only on the basis of performance, but on degree of trust, on the community, on the kind of service request, and other metrics. Application level behavior may draw upon this reputation among other mechanisms. For example, a user might trust anyone to forward his messages, but only accept address book update from identities whose reputation exceeds a certain threshold, or from nodes that have forwarded its service requests in the past.

[4]Trust is the belief that a person is who his identity claims he is.

Therefore, reputation systems will be a strong components (not necessarily the only) in creating an incentive to cooperate. Encouraging users to contribute resource (memory, battery, time, etc.) will be a major problem in HAGGLE. The problem is solved in Kazaa by limiting the amount of information that is made available to those that do not want to contribute, but just consume. We can implement the same kind of mechanism in HAGGLE. However, we believe that communities will be a strong element in getting a given user more cooperative.

Note that the reputation system is just one component of an incentive to cooperate mechanism.

We identify the following open problems:

Incentives vs Reputations Without special hardware, true incentives are difficult. How paranoid can we be without disabling useful functionality?

decentralized identity and token checking

4.3 Additional HAGGLE Functionalities

We now discuss additional components that need to be provided by HAGGLE in order to support the largest possible set of applications.

4.3.1 Localization

Most HAGGLE applications will rely on locality information such as geographical location, or neighborhood. However, localization does not necessarily mean GPS. There are numerous localization algorithm that could be implemented in HAGGLE. Each of them will match specific community needs, and have an impact on the way services can be provided and service requests are forwarded.

We identify the following open problems:

absolute vs. relative localization

4.3.2 Forwarding Information Management

HAGGLE nodes rely on locally stored information to make a forwarding decision for each service request. The information available can have multiple source:

- Users information such as address book, local files, applications,
- Network interfaces available in the device and communication protocols.
- Historical data.
- Community specific information

It is important to define how each dataset can be utilized and shared, but also how it can be stored and accessed.

We identify the following open problems:

local database management

aging

4.3.3 Legacy Application Interaction

The HAGGLE information space concept can encompass support for legacy applications. Email services are modeled by having the HAGGLE forwarding engine regard a recipient's email server as a high-priority next-hop, and having a user's device issue queries against that information space periodically.

Web page requests can be treated as service requests, with the page itself being a HAGGLE data item with metadata indicating the recipient. A node with Internet access would transform the HAGGLE request into a suitable web search and send back a HAGGLE message with the resulting objects. Web caching may be automatically accomplished by nodes keeping copies of responses to recent queries, if they have space.

We identify the following open problems:

Natural Fit Email, news, etc. map well onto HAGGLE.

End-to-End Users will have to tolerate some delay (mitigated by HAGGLE caches) when accessing traditional services.

4.3.4 User Interface

In the current networking world, users are often forced to make routing decisions when trying to send data to local recipients, e.g. having to pick one of email, infrared transfer, Bluetooth transfer, USB key, or another method when wishing to transfer files to other local recipients — and the list keeps growing. This is precisely opposed to the goals of transparency and ease of use which are held dear to computer users. In the HAGGLE, we may be able to offer the advantage that both local and wide-area connectivity are made transparent to users.

However, transparency is not necessarily the only goal. It is important that users remain appraised of the status of the delivery of the data they send and request. With end to end communications, this is relatively simple to determine and to display for the users convenience, e.g. using a spinning globe icon in a web browser, which stops

spinning when page-load is complete. In a HAGGLE, not only are there many more states which one may wish to communicate, but a HAGGLE client may also have little indication of the network status, since some nodes it was communicating with are no longer visible. Allowing HAGGLE users to achieve and maintain an intuitive mental model of the status of HAGGLE may be a key issue in providing usable and deployable HAGGLE apps.

Explicit user involvement in certain situations is also necessary, for example in determining a trust relationship if there is no prior community network of trust to draw upon, or in mapping a device's public key to a user that it claims to represent (similar to the way `ssh` maps keys to hosts). Users should also retain control over all HAGGLE operations since they may involve the spending of scarce resources such as storage, bandwidth or battery.

We identify the following open problems:

Transparency Users should have to make as few decisions as possible to achieve a desired result.

Feedback Useful feedback about the state of the world needs to be conveyed to the user. we might have a tuple in the haggle messages that explicitly ask for feedback from intermediate nodes and have a way to summarize the feedback that would be close to a progress bar.

4.3.5 Monitoring

A monitoring module will collect traffic information to make it possible to analyze a number of "dispatching" or "forwarding" strategies, applications behaviour and so forth.

4.3.6 Applications

HAGGLE should allow legacy application to operate naturally in an opportunistic environment. HAGGLE should also make it possible to deploy new application, out of which the next "killer app" might emerge.

5 HAGGLE architecture principles

We now introduce Haggle an architecture. We intend that this architecture will provide the basis for a wide variety of OppNet applications. We expect that the full specification and implementation of Haggle will take several years. Development will involve a process of iterative refinement, driven by feedback from the deployment of trial applications. More details about Haggle and its design can be found in a companion technical report [7].

5.1 Application-level Data Units

In the spirit of "Application Layer Framing" proposed by Clark and Tennenhouse in the early nineties [5], the Haggle architecture is not layered; application-layer and network-layer information are collapsed into one space. We represent messages as application-level data units (ADUs), which are comprised of a number of attribute-value pairs.

A node decides whether to offer an ADUs to a neighbour based on many attribute-value pairs, rather than simply on the destination address as in IP-centric networking. This is useful because, *in Haggle, an ADU's destination can be loosely specified, or not specified at all*. A sender may not know the set of devices available to the recipient. Alternatively, for message types such as information queries, many devices could take the role of destination.

A node might accept transmission of an ADU from a neighbour for two reasons. One is that the node may determine that it is a valid recipient, since it might carry an application which is interested in the ADU. The other is that the node might decide that it has a high expectation of further forwarding opportunities for that ADU, for example because the recipient was recently seen by it, or because the node expects to acquire global connectivity shortly, and can thereby deliver the message. Again, the visibility of application-level information in various attribute-value pairs may usefully contribute to a node's decision.

Examples of possible attribute-value pairs for various Haggle applications are:

```
message-type: private message
originator:NAME James Scott
sender-authentication: [cryptographic signature]
recipient: NAME Eben Upton
recipient: SMTP eben.upton@intel.com
recipient: SMS +451233764432
encrypted-data: [encrypted set of tuples with messa

message-type: query
data: opportunistic networking

message-type: public message
forum: recipes
subject: chicken madras
```

5.2 Node Architecture

The preliminary design of the Haggle node architecture, illustrated in Figure 1, is based around the notion of

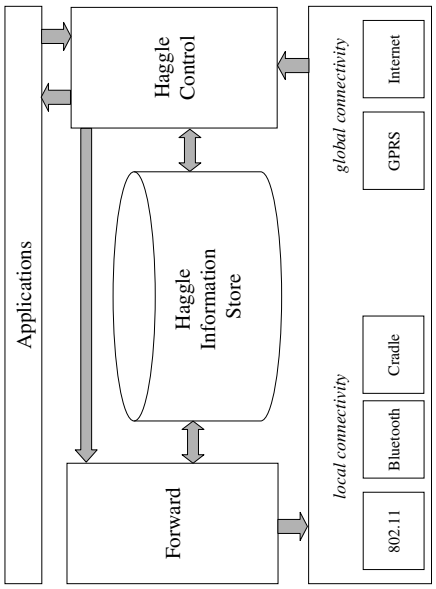

Figure 1: Prototype Haggle architecture.

a *Haggle Information Store* (HIS) containing ADUs on each user device. Applications can inject new ADUs into the HIS, can register interest in incoming ADUs by specifying matching criteria for particular attribute-value pairs, and can search the HIS for existing ADUs.

The *Haggle control* module decides whether to accept an incoming ADU into the HIS, provides an API for applications, and controls the forwarding module. The *forwarding* module performs neighbour discovery using the available network interfaces, and when necessary connects to neighbours which are identified as potential next-hops for ADUs in order to offer those ADUs for transmission. Note that, where it is available, the Internet is regarded as just one of many network interface types.

6 Implementation

We start with a discussion of the implementation challenges. Note that HAGGLE architecture is going to be implemented and evaluated with real users. We are not yet close to prove the feasibility of HAGGLE. Therefore, we address the HAGGLE feasibility under three orthogonal angles.

- Identify implementation challenges.
- Study the feasibility of Opportunistic Networking.
- Implement a case study with cell phones, PDAs, and laptops using the BlueTooth communication technology.

6.1 Implementation Challenges

We have identified a number of key challenges which Haggle must overcome if it is to become widely used.

6.1.1 Device Resources

Haggle makes demands on a number of device resources. Substantial processing power is required to perform encryption and decryption of messages. Storage is required for the HIS; we anticipate that maintaining a large number of 'in transit' ADUs in the HIS will improve the forwarding success rate.

Mid-priced smartphones today have 100MHz RISC processors, Just in Time (JIT) compilers, and tens of megabytes of available storage. As both processing power and available storage are increasing rapidly, we are confident that this challenge may be overcome.

6.1.2 Bandwidth

Network bandwidth is another scarce resource. Scarcity derives from the limited channel capacity of the wireless technologies used, and from contention between the potentially large numbers of devices in range. Adaptive throttling of transmission rates at the application level may prove necessary.

6.1.3 Power

Another limitation is available battery capacity. This is growing at a more modest rate than, for example, processing power, and may therefore emerge as the primary limiting factor in future devices. On multipurpose devices (smartphones and PDAs), the Haggle application must ensure that power is remains available for the device's primary function; users will cease to use Haggle if it drains their batteries by midday.

To provide an incentive for mobile users to expend this valuable resource for the benefit of other users, we will have to rely on communities (as described above), as well as on straightforward social etiquette.

6.1.4 Link Layers

Different devices will support different sets of radio interfaces. Where supported, we will provide the ability to transmit ADUs over Bluetooth, 802.11 and GSM radios. We need to determine to whether Bluetooth and 802.11 (which share the 2.4GHz band) can be used together, or whether we must switch between the two.

6.2 Feasibility of Opportunistic Networking

As described in Section **??**, transfer opportunities are time-limited, and existing technologies and protocols are not optimised for this case. Nonetheless, devices supporting Bluetooth and 802.11 are widely deployed, and their numbers are expected to increase rapidly in the future [3]. In this section, we present a preliminary set of measurements and simulations of Bluetooth data transfer.

The amount of data that can be transferred between two mobile nodes that encounter one another is dependent upon several factors: the time required for the nodes to discover one another, the time that the nodes are within radio range of one another, and the variation of throughput with range and operating environment. We measured each of these factors, and used the results to simulate a series of transfer opportunities, calculating the amount of data that one would expect to be transferred in each case.[5]

We performed our measurements using the PC laptops running Windows XP and class II Bluetooth USB devices manufactured by MSI and Belkin. We observed no significant performance differences between the two varieties of device. The measurements presented here were obtained using the MSI devices, and were taken one meter above the ground. All other wireless devices in the machines were removed.

We first measured the distance at which the devices could reliably discover one another's presence (*inquiry*), and determined this to be approximately 50 meters. The inquiry process requires that the inquiring node place its radio into a mode that renders it invisible to other inquiries. As there is no central coordination between nodes, we measured the probability of a successful inquiry when the inquiry period of two nodes overlap. We observed that a non-overlapping period of approximately 3.5 seconds was enough to reliably establish a connection.

The goodput between devices was measured at various distances by opening an RFCOMM connection between the machines and sending 64 kilobyte messages from the initiator to the slave. We performed this measurement in two environments: indoors, in an office corridor in the presence of background 802.11 and Bluetooth interference, and outdoors, in a field far from such interference. As seen in Fig. 2, there was significant variability in the results, particularly around 2.5 and 6 meters. Further, performance was better indoors than out. We believe that some combination of factors including radio propagation variability, multipath interference, and external interference are responsible for this variability.

[5] We also performed a mathematical analysis of such a scenario. Due to its length we omit it here, but we remark that it broadly agrees with our measurements and simulation.

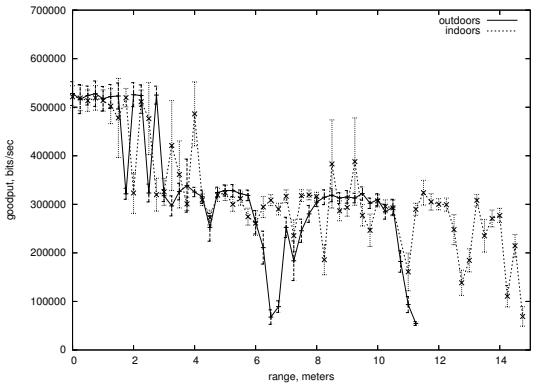

Figure 2: Bluetooth RFCOMM goodput.

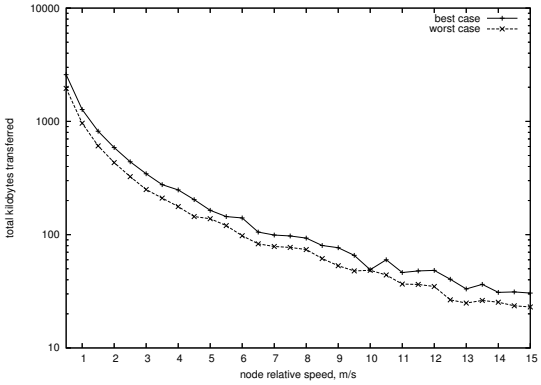

Figure 3: Expected number of kilobytes sent during a transfer opportunity.

Finally, we created a simulation of limited-duration transfer opportunities. In the simulation, two nodes performing inquiry approach one another head on, pass at some relative speed, and eventually move out of range. Once inquiry is successful, one node sends data until out of radio range. The average number of kilobytes transferred (in 50,000 experiments at each point) in two cases is shown in Fig. 3. The solid line indicates the results obtained using the best case values from Fig. 2. The dashed line indicates the results obtained using the worst case values.

Although these results are preliminary, they indicate that Bluetooth is usable for opportunistic networking. In

future work, we plan to characterise the factors responsible for the high observed variation in goodput. We also plan to examine the effects of battery life and interaction with higher-level protocols.

6.3 Implementing HAGGLE on Bluetooth

The reference implementation of Haggle is written in Java. It runs on Pocket PC devices under the IBM J9 JVM, and on Windows XP PCs under the Sun JVM. We use our own version of the JSR 82 Bluetooth API (Blue Cove), built on top of the WIDCOMM and Microsoft Bluetooth stacks, for communication between devices. Here we present an outline of the features of our prototype implementation.

6.3.1 Blue Cove

The JSR 82 standard specifies a Bluetooth API for use as an optional component of the Java 2 Micro Edition (J2ME) platform. As there is no pre-existing free version of this API for Pocket PC and Windows XP machines, we have developed our own implementation, Blue Cove. This is available under an open-source license from source forge[6]. Blue Cove supports a large subset of JSR 82, and runs on all our target platforms.

By developing against a standard API, we ensure that Haggle will run on smartphones (such as the Nokia 6230 and 6600) which support Bluetooth.

6.3.2 Application Data Units

The Application Data Unit (ADU) is the basic unit of information storage in the Haggle architecture. Each ADU is a map from attributes to values. Both attributes and values are blobs (blocks of binary data), although by convention attributes are restricted to human-readable text strings expressed using the UTF-8 encoding scheme.

We define a small number of attributes which are meaningful to the Haggle infrastructure. These are

from: string message source address *to: string* message destination address *id: string* sender-allocated message identifier

ADUs may be serialised to, and loaded from, persistent storage. At present all values in loaded ADUs reside in memory. We are in the process of adding support for values which transparently reside in persistent storage. This feature will be essential for file sharing applications.

A *digest* of an ADU is a subset of the attribute value pairs contained in the ADU. ADUs are represented by instances of the mutable Java class ADU. Attributes and values are represented by instances of the immutable class *Blob*.

[6] http://www.sourceforge.net/projects/bluecove

6.3.3 Haggle Information Store

Each device possesses a Haggle Information Store (HIS), in which it stores its local data. The HIS is a bag of ADUs. Applications use the Haggle API to iterate over the contents of the HIS, to insert and remove ADUs, and to receive notification when new ADUs arrive from the network.

The contents of the HIS are loaded from persistent storage when a Haggle application starts, and serialised to persistent storage when it ends. The HIS is represented by a singleton instance of the Java class *HIS*.

6.3.4 Forwarding over Bluetooth

Bluetooth is a connection-oriented wireless protocol intended primarily for cable replacement (a successor to IrDA). It is supported by many recent laptop PCs, PDAs and smartphones; projections indicate that there will be in excess of 100 million Bluetooth devices by 2007.

The low-level forwarding scheme used by our reference implementation cannot use broadcast (as would be sensible under 802.11). Instead we establish an individual point-to-point connection with each visible device in turn.

Every Bluetooth device possesses a unique 48-bit address. A Bluetooth device can discover the addresses of its neighbours using *inquiry*. While it is performing inquiry, a device is invisible to its neighbours. Determining how frequently to inquire so as to minimise the expected time to discover a new neighbour is the subject of ongoing research at IRC. Each Haggle node repeatedly performs inquiry and attempts to connect to neighbouring devices. We choose the inter-inquiry delay according to an exponential distribution with a mean of 10 seconds.

A Bluetooth device typically offers one or more services, which are described using 128-bit UUIDs. Connection attempts specify the UUID of the target service. A device can discover the services offered by another device using *service discovery*. Each Haggle node advertises the Haggle service, and accepts at most one incoming connection at a time. This limit is imposed by the Bluetooth hardware. Bluetooth connections are reliable, bi-directional streams.

Upon establishing a connection, the connecting device transmits digests of a set of ADUs taken from its HIS. If, on the basis of the information contained in the digest, the connectee wishes to receive a particular ADU, it can request the remaining attribute-value pairs. The process is

then repeated, with the connectee offering digests to the connecting device.

6.3.5 Policies

There are several aspects of the forwarding policy which may be adjusted

- Which ADUs do we offer to a given device?
- Which attribute-value pairs from a given ADU do we include in a digest?
- Do we accept a given ADU from a given device on the basis of a digest?

These decisions are abstracted by a Java interface *Policy*. At present, we provide a placeholder implementation which

- Offers all ADUs to all devices.
- Includes a statically-defined subset of attributes (*from, to, id*) in each digest.
- Accepts all previously unseen ADUs.

This implementation corresponds to the flood forwarding algorithm. As our understanding of forwarding develops, we will be able to provide more efficient implementations. It is anticipated that application-specific information will play an important role, as illustrated in the examples below.

6.3.6 User Interface

We provide Java interfaces *ADUListener* and *InfoListener*, which applications may implement in order to receive notification when new ADUs arrive and when certain events occur within the Haggle system (discovery of neighbouring devices, incoming and outgoing connections).

We also provide Standard Widget Toolkit (SWT) components *ADUView* and *Console* which implement these interfaces. Applications may instantiate these components and use them as part of their own user interfaces.

7 Application

7.1 Haggle News

The Haggle News application provides distributed, threaded, Usenet-style newsgroups. Postings are represented as ADUs having the following attributes.

subject: string posting subject line *body: string* posting body *message-id: int* message identifier *parent-id: int* message identifier of parent message, not present for root messages

At each connection attempt, the each party offers the other up to 100 messages. Devices keep track of which messages they have offered (or delivered) to other devices to avoid duplication of effort.

Digests contain only the *message-id* and *parent-id* attributes. Although we could just transmit *message-id*, including *parent-id* allows the receiver to prioritise a message whose parent is already known over a disconnected thread fragment.

The ADU acceptance policy either accepts all previously unseen messages, or prioritises as described above.

7.2 Content sharing

We now describe an implementation of the content sharing application motivated in section 2.1, in the context of the HAGGLE infrastructure.

Cached content in our application is represented by tuples with the following attributes

`content_type` A MIME type [] indicating the kind of content stored in the tuple

`content` The binary content data itself

`content_metadata` A comma-separated list of terms describing the content

Queries are represented by tuples with the attributes `target` and `ttl` (required by the forwarding system), and the following attributes

`query_types` (optional) A comma-separated list of MIME types indicating the kind of contents which the requester is prepared to accept

`query_metadata` A comma-separated list of terms describing the required content

`source` The name (see section 4.2) of the requester.

Upon receiving a tuple which describes a query, the application examines the cache to find an item of content whose `content_type` matches one of the elements of `query_types`, and whose `content_metadata` contains all of the elements of `query_metadata`.

If a matching item is found, a copy is injected into the cache, with added attributes `target` (set to the `source`

of the query) and `ttl`. This copy is now available for forwarding.

Upon receiving an item of content, the requester removes the `target` and `ttl`. The item is therefore no longer available for routing.

We provide a user interface to make the operations of sharing and searching for files as intuitive as possible.

8 Deployment Plans

We are in the process of extending HAGGLE to utilize other wireless data bearers (such as Wifi and GPRS), and to make more extensive use of wired networks where they are available. Ports HAGGLE to new systems are relatively simple given the library code we have written thus far. The major issues revolve around proper design of user interface for systems with different input and output mechanisms.

We are developing a range of client applications to enable the services described in section 2.1, including asynchronous messaging, a distributed address book, and file searching and sharing.

Finally, we are conducting simulations of the system, using random waypoint models and a small corpus of real-world data. We wish to ensure that the system is sufficiently scalable to function correctly during a medium scale (100+ nodes) trial deployment in early 2005.

References

[1] http://moment.cs.ucsb.edu/AODV/aodv.html#Implementations.

[2] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. Technical Report 96-17, AT&T Research, 1996.

[3] http://www.bluetooth.com/.

[4] Marc Branchaud. A survey of public-key infrastructures. Master's thesis, McGill University, 2997.

[5] D. Clark and D. Tennenhouse. Architectural considerations for a new generation of protocols. In *ACM SIGCOMM*, pages 200–208, September 1990.

[6] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46, 2001.

[7] Christophe Diot, James Scott, Eben Upton, and Marc Liberatore. The Haggle architecture. Technical Report IRC-TR-04-016, Intel Research Cambridge, 2004.

[8] http://www.dtnrg.org/.

[9] K. Fall. A delay-tolerant network architecture for challenged internets. In *ACM SIGCOMM*, August 2003.

[10] Gavin Holland and Nitin Vaidya. Analysis of TCP performance over mobile ad hoc networks. In *5th Annual ACM/IEEE international Conference on Mobile Computing and Networking*, pages 219–230, 1999.

[11] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[12] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, San Jose, CA, October 2002.

[13] Tim Kindberg and Kan Zhang. Validating and securing spontaneous associations between wireless devices. In *Proceedings 6th Information Security Conference (ISC 03)*, 2003.

[14] Jinyang Li, Charles Blake, Douglas S. J. De Couto, Hu Imm Lee, and Robert Morris. Capacity of ad hoc wireless networks. In *Proceedings of the 7th ACM International Conference on Mobile Computing and Networking*, pages 61–69, Rome, Italy, Jul 2001.

[15] Stanley Milgram. The small world problem. *Psychology Today*, 61:60–67, may 1967.

[16] Maria Papadopouli and Henning Schulzrinne. Performance of data dissemination among mobile devices. Technical Report 005, Columbia University, 2001.

[17] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[18] Charles E. Perkins and Elizabeth M. Royer. Ad hoc on-demand distance vector routing. In *IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[19] E. Royer and C. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications*, April 1999.

[20] C. Tschudin, H. Lundgren, and E. Nordstrom. Embedding MANETs in the real world. In *8th IFIP International Conference on Personal Wireless Communications (PWC2003)*, sep 2003.