Haggle: Seamless Networking for Mobile Applications

Jing Su[1 2], James Scott[1 5], Pan Hui[1 3 4], Jon Crowcroft[3], Eyal de Lara[2]
Christophe Diot[4], Ashvin Goel[2], Meng How Lim[1], and Eben Upton[1]

[1] Intel Research Cambridge
[2] University of Toronto
[3] Cambridge University
[4] Thomson

Abstract. This paper presents Haggle, an architecture for mobile devices that enables seamless network connectivity and application functionality in dynamic mobile environments. Current applications must contain significant network binding and protocol logic, which makes them inflexible to the dynamic networking environments facing mobile devices. Haggle allows separating application logic from transport bindings so that applications can be communication agnostic. Internally, the Haggle framework provides a mechanism for late-binding interfaces, names, protocols, and resources for network communication. This separation allows applications to easily utilize multiple communication modes and methods across infrastructure and infrastructure-less environments. We provide a prototype implementation of the Haggle framework and evaluate it by demonstrating support for two existing legacy applications, email and web browsing. Haggle makes it possible for these applications to seamlessly utilize mobile networking opportunities both with and without infrastructure.

1 Introduction

Advances in computing technology have had a profound impact on the capabilities of portable devices such as smart-phones, notebooks, and personal digital assistants. Today these devices provide a rich computing environment and multiple communication methods based on different radio technologies such as short-range Bluetooth, medium-range 802.11 and longer-range cellular radios.

Users expect ubiquitous access to applications such as messaging and information browsing on these powerful devices. Unfortunately existing applications are often unable to take advantage of the mobility and connectivity options that may be present because they are written to a software abstraction that is deeply intertwined with the underlying networking architecture. This is illustrated by the fact that applications must currently be responsible for establishing all bindings necessary to perform communication. This requirement causes applications to assume the implicit design, conventions, and operating modes of the underlying networking. As a result, applications are difficult to adapt to new communication mechanisms. For example, email and web addresses implicitly assume a naming structure which requires the use of highly available DNS servers.

[5] now at Microsoft Research Cambridge

We believe that the user experience should be that of applications adapting to changing network conditions with devices responsive to different available connectivity options, protocols, and communication environments. For instance, email should be sent peer-to-peer if the sender and recipient are in close proximity, browsers should be able to search neighbours for possible matching content if the Internet is not reachable, and devices should be able to utilize the connectivity of peers willing to provide a bridge to the Internet. Currently, it is non-trivial to add the decision logic into applications to handle these different usage models, and furthermore, if each application makes the decisions individually, the overall system may perform poorly [4, 5].

In this paper, we present Haggle, a novel framework that enables seamless network connectivity and application functionality in dynamic mobile environments. At its core, Haggle allows separating application logic from the underlying networking technology. Applications delegate the task of handling and communicating data to Haggle, which in turn adapts to the current network environment using the best available connectivity and protocol for the situation and user-specified policies that allow trading speed, cost and power constraints.

Haggle employs three main ideas to achieve its goals (Section 3). First, it is able to adapt to its mobile environment by delaying network connectivity interface and protocol selection until the moment of data transmission, a technique also known as late-binding. This approach allows operating across multiple interfaces (possibly concurrently), protocols and applications. Second, applications can share their data and meta-data with Haggle, which allows localized data search and sharing, such as browsers being able to search neighbours for matching content. Third, Haggle provides a unified mechanism for managing these shared data and network resources centrally on each device, according to user preferences and expectations.

These Haggle concepts are realized in an architecture for which we have developed a freely available open-source prototype implementation (Section 4). The prototype currently supports existing email and web applications, allowing them to seamlessly operate across environments with and without infrastructure (Section 5). We also present experimental evaluations demonstrating Haggle's ability to allow these applications to function seamlessly in highly dynamic mobile environments (Section 6).

2 Motivation

We begin with a motivating example for the problem with current networking state and why it is lacking. Alice and Bob are in a train heading towards the city. Alice wishes to forward Bob a discussion thread containing a document for review. However, the email may be difficult to send due to absence of any Internet connectivity, or slow and expensive to send as the mail is sent over a cellular link to an email server and then retrieved from the server by Bob's device.

For users Alice and Bob it is not intuitively obvious why it is so difficult or slow to send the email and attachment. Ideally the contents should be sent over a fast mutually supported peer-to-peer technology such as Bluetooth or 802.11 in ad hoc mode. However, even if Alice and Bob mutually configured an ad hoc network between their devices, their email programs would still not be able to communicate across this ad hoc network since email protocols assume the presence of infrastructure services.

Our second motivating example considers Charlie who wants to read some news to pass time in the train, but is either outside of the cellular coverage area or subject to high roaming charges. Currently, Charlie would not even try to use his web browser to read news since he knows there is no connection. However, since reading news is a popular activity in the train, it is highly likely that other people around him will have some reasonably matching cached content available. Unfortunately this information is not available to him.

We observe that the current networking framework is not flexible enough to support applications and mobile users in an intuitive and simple way. The problem is that we need a smart method for selecting the connectivity method, protocols and name bindings appropriate for the connectivity method, and a mechanism for managing the device's communication resources across the various applications on a device. For example, in the case of Alice and Bob, the non-intuitive reason why their email programs won't communicate over an ad hoc link with each other is because email clients assume the availability of DNS services to look up MX records for the domain portion of email addresses, and expect to contact the mail server found in the MX record – both of which are not available in a local ad hoc network. Our goal in Haggle is to provide a networking framework for applications and users that enables these usage models and provides an intuitive mechanism for specifying policies and preferences.

3 The Haggle Approach

The key insight in Haggle's approach is that applications should not have to concern themselves with the mechanisms of transporting data to the right place. Instead, this should be left to the networking architecture. Providing this separation of concerns not only simplifies the application logic, but allows them to automatically adapt to new mobile environments and technologies. To achieve this separation, Haggle uses a data-centric network architecture [2] that internally manages the task of handling and propagating data. Applications can then be automatically adapted to dynamic network environments using the best connectivity channel for the situation. Below, we identify three principles that are critical to our approach.

Just-in-time Binding of Interfaces, Protocols, and Names: At the hardware interface level, mobile devices provide different (and often multiple) connectivity interfaces of varying characteristics depending on their intended usage situations. Networking technologies can differ in many aspects, ranging in physical characteristics such as power and range, communication characteristics such as latency and cost, and peering characteristics such as ad hoc and neighbour discovery. We aim to support and embrace the use of many different networking technologies at the same time. To achieve this flexibility, we use late-binding (or just-in-time binding) of the connectivity interface to use, balancing the interface characteristics with user and application supplied preferences.

In mobile environments, depending on the interface that is selected and the connection context, the necessary protocols for performing networking operations can vary.

For example, the SMTP protocol is needed for sending mail messages to a server, but a peer-to-peer protocol should be used for sending to the recipient in close proximity. Haggle allows supporting multiple routing protocols such as peer-to-peer and intentional naming [1], and late binding to these protocols in different environments to automatically adapt applications.

In order to identify services, devices, or individuals, it is necessary to support a naming system that is flexible enough for the different networking environments. Specifically, it is not possible to resolve DNS names in ad hoc environments when infrastructure is absent. Furthermore, different entities along the delivery path may have different name resolution mechanisms. Thus it is necessary to have a flexible and semantically rich naming system which can support late-binding specification of services, individuals, or devices.

While late binding has been explored in several contexts [1, 22], Haggle is unique in allowing late binding at the three levels described above.

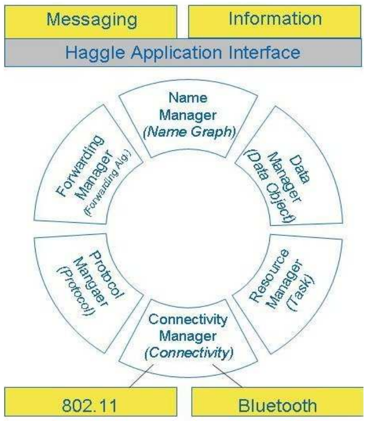
Exposure of Persistent Data and Metadata: To facilitate the correct searching, sharing, and opportunistic use of data, it is necessary to employ the help of applications since much of the metadata context is embedded in the application logic. For example, in order to answer queries for keyword-matching web pages in the local cache, we must have metadata for the browser's cache of pages, images, related links, and relative freshness.

Providing support for data-driven network operations requires exposure of data and metadata context outside of the application logic. We provide two classes of metadata: attribute tags and relationships. Data objects can be tagged with arbitrary sets of attribute key/value pairs, and relationships between objects can be established using directed edges. Relationships can have many different semantic meanings, established by attribute values on the edges themselves. In this paper, we consider two distinct relationships: ownership and dependency. We elaborate on the details of our use of these two relationships when we describe, in Section 5, the applications we use in our experiments.

Centralized Resource Management: Haggle manages the use of networking resources on the device centrally to ensure that the behaviour of the mobile device conforms to the expectations and preferences of the user. On a user's mobile device, there may be many applications running, each with varying types of simultaneous requests. Assuming that applications are "smart" and internally support all of the features described above, it is likely that the selfish actions of some applications will result in poor and unmanageable system behaviour. In Haggle, all requests for manageable resources are issued as *tasks* to a centralized module which dictates which actions are allowed to proceed, in accordance with current context and user-specified policies.

4 Haggle Architecture and Prototype

Haggle is internally composed of six managers organized in a layerless fashion (Figure 1) in contrast to the stacked approach of TCP/IP. The managers are each responsible for a key modular component or data structure (shown in italics in the diagram) - this provides flexibility e.g. allowing for many protocols (e.g. SMTP and HTTP) and connectivities (e.g. 802.11 and Bluetooth) to be instantiated simultaneously. The managers and modules all have well-defined APIs, and each manager (and internal component) may use the API of any or all of the other managers. This novel architecture provides necessary and useful flexibility over a layered architecture in which each layer may only talk to the two APIs above and below.


Messaging
Information
Haggle Application Interface
Name Manager (Name Graph)
Data Manager (Data Object)
Resource Manager (Task)
Connectivity Manager (Connectivity)
Protocol Manager (Protocol)
Forwarding Manager (Forwarding Alg.)
802.11
Bluetooth
Fig. 1: Architecture overview

Externally, the application layer API is a subset of the interfaces provided by the individual modules. For more details, please see the technical report [23]. The remainder of this section describes how this architecture supports the core concepts of just-in-time binding, data management and resource scheduling.

4.1 Just-In-Time Binding

Connectivity Interfaces: Haggle aims to embrace the use of many different networking technologies at the same time. Networking technologies can differ in many aspects, including range, latency, bandwidth, cost, availability, power, and so on. It is therefore appropriate for different connectivity interfaces to be used depending on the particular type of data being sent.

For each network interface on a node, we construct a connectivity instance. For example if there were two 802.11 interfaces there would be two connectivity objects, one for each interface. This is because a connectivity is regarded as a schedulable resource which can consume network time, battery power, monetary costs, etc. As a scheduled resource, all operations that result in network activity, including operations initiated by the connectivity itself, must be delegated for scheduling.

Connectivity objects in Haggle must support a well-defined interface including functionality for neighbour discovery, opening/using/closing communications channels, and estimating the costs (in terms of money, time and energy) of performing network operations. Each connectivity must interface with the underlying driver and hardware to provide this functionality.

Neighbour discovery can take various forms, depending on the connectivity. In 802.11, any node with reception turned on can see beacons from access points which announce their existence. For Bluetooth, neighbour discovery is an active (and time-consuming) process. For GPRS, neighbour discovery is implicit in that when base station coverage is present the Internet is accessible. Delegating the initiation of such operations is an important design approach which enables Haggle to manage multiple interfaces with respect to user defined policies.

The prototype implementation of Haggle focuses on using the 802.11 connectivity because it is a widely used wireless access network and is available for a range of devices from laptops to mobile phones. It also offers both neighbourhood and infrastructure connections (through ad hoc mode and infrastructure mode respectively) which allow us to explore the range of Haggle capabilities using a single connectivity type.

As a schedulable network resource, 802.11 interfaces must provide a cost function, which we currently model in terms of time-on-network cost. For data transfers, the time-on-network cost is calculated per byte, taking into account the bandwidth and size of data. We used a lower bandwidth estimate for AP mode than ad hoc mode since we expect the access link to be the bottleneck in AP mode. This would ideally be dynamically measured on a per-AP basis rather than statically estimated. When switching to AP mode from ad hoc, there can be a delay of a number of seconds due to the latency of DHCP to provide an IP address. We model this as a 5 second switching overhead, which we have experimentally determined to be fairly typical.

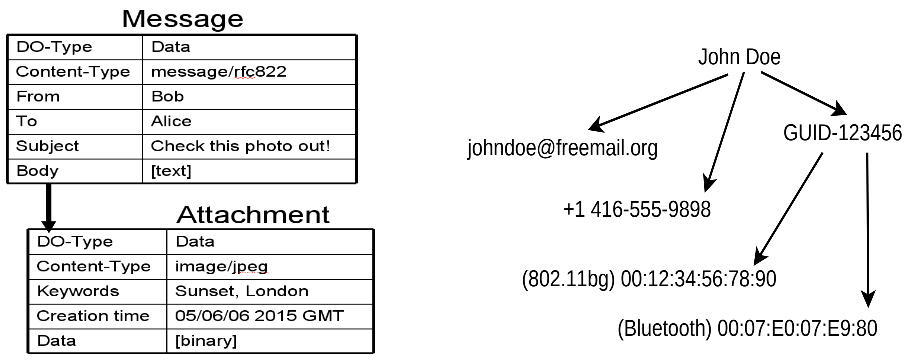
Protocols and Forwarding: Haggle encapsulates the late-binding of communication protocols and forwarding algorithms necessary for transporting data. Communication protocols specify the method for point-to-point communication, both for transmitting data as well as opening and receiving connections. For example, the HTTP protocol specifies how to connect to a web server and request objects, while the peer-to-peer protocol specifies how to connect to and receive connections from peers.

When connecting to peer or infrastructure endpoints, the most appropriate protocol is selected just-in-time to perform the necessary initializations as well as transformations and translations in order to send and receive. For protocols which must accept incoming connections, such as a peer-to-peer protocol, the protocol must provide sufficient information to the connectivity interface so that incoming connections can be redirected and properly handled by the protocol.

Forwarding algorithms determine the suitability of a next hop for transmission of application and user-level messages. The suitability is presented as a benefit value which enables Haggle to select the just-in-time binding for the forwarding algorithm, communication protocol and connectivity interface. Forwarding algorithms can be active entities, generating and receiving network messages required for maintenance and routing in an overlay or ad hoc network. Such messages, like all network use operations, must also be delegated for scheduling.

Haggle's flexible architecture allows many forwarding algorithms to be in use *simultaneously*. Possible algorithms can range from epidemic [24] to MANET algorithms such as geographic [15] or distance-vector [16], as well as store-and-forward [20, 26] or mobility based [10, 13, 14]. Delegating the proposed actions essentially allows the forwarding algorithms to compete for action. The most applicable algorithm for a given environment will prevail.

We implemented two forwarding algorithms so far, namely "direct" and "epidemic". The direct algorithm only proposes to deliver messages to their destinations if the destination is reachable by direct communication. As a result, the direct forwarding


Message

DO-Type	Data
Content-Type	message/rfc822
From	Bob
To	Alice
Subject	Check this photo out!
Body	[text]

Attachment

DO-Type	Data
Content-Type	image/jpeg
Keywords	Sunset, London
Creation time	05/06/06 2015 GMT
Data	[binary]

(a) Message and Attachment (b) Name Graph

Fig. 2: Example Data and Name Object Graphs

algorithm will always propose a delivery benefit of 100%. The epidemic algorithm proposes to send messages to all immediately reachable destinations. Because the epidemic algorithm cannot be certain if flooding will reach the destination, it will propose a lower delivery benefit value.

Names: Current networking architectures require early-binding of names as nested headers found in the front of physical-layer packets. Current dynamic name resolution systems such as DNS still require eager resolution of human-readable names to routable addresses, which then must be bound to the physical address of the transmission interface. Unfortunately this paradigm does not work well in mobile environments where infrastructure services such as DNS might not be available at the moment of the application's request, or the connectivity interface has different resolution semantics such as Bluetooth discovery.

Haggle presents a general form of naming notation that allows late-binding of many user-level names, independent of the lower-level addressable name, as proposed in i3 [22]. We achieve this by using *name graphs*, inspired by INS [1], which are hierarchical descriptions of many known mappings from a user-level endpoint to lower-level names (which may imply particular protocols/connectivity methods). Name graphs are used as recipient identifiers for messages as well as identifying the source node and any intermediate nodes. This late-binding approach contrasts with the existing network architecture were names are only meaningful at particular layers of the protocol stack.

What's in a name? An example name graph, which illustrates the provision of many deliverable addresses for communication endpoints is shown in Figure 2(b). The figure illustrates how one individual, John Doe, can have many different addressable identities, reachable using different connectivity methods. Name graphs span from top-level nodes such as personal names through to leaves comprising persistent methods of reaching them, such as email addresses, but not transient addressing data such as the IP address for the email server. The choice of this partition [12] stems from the feature that any "name" in Haggle can also be an "address" if there exists a suitable protocol which understands that name. For example, an SMS-capable device regards a phone number name as an address, but a non-SMS capable device would not. As a message moves between nodes, different methods of mapping names to transmission methods can become available. Transient names such as looked-up IP addresses are not valid "names" since they do not provide useful identity information for another node.

Haggle's design allows it to take advantage of any number of existing name management schemes that have been explored in previous work, such as Persistent and Personal Names [9]. In the prototype, we utilize a hierarchical name graph construction. On first startup, Haggle nodes create a GUID name to identify the node itself. Then, the MAC addresses of the node's interfaces are also created as names under the root name. New names can also be learned from applications. For example the names and emails of the sender can be captured from an outbound email and added to the name graph as part of the person's identity. This identity graph is then linked to the node graph to indicate ownership of the device by the individual.

In the prototype implementation, Haggle nodes actively request tasks to contact newly visible names which have no associated node or user identity information. If the peer responds with identity information, it is merged into node's knowledge-base of names. In this way peer nodes can learn identity information even if infrastructure is not available.

4.2 Data Management and Data Objects

Haggle exports an interface for applications to manage persistent data and metadata explicitly. Haggle's data format is designed around the need to be *structured* and *searchable*. In other words, relationships between application data units (for example, a webpage and its embedded images) should be representable in Haggle, and applications should be able to search both locally and remotely for data objects matching particular useful characteristics. We draw inspiration from desktop search products (e.g. Google Desktop) which have changed the way that many users file and access their data [7], allowing us to avoid having to methodically place data in a directory structure. We propose that applications can use a combination of structured data and search, with the former providing the kind of capabilities expected of a traditional file-system, and the latter allowing applications to easily find and use data that they themselves did not store.

Data Objects: A Data Object comprises many *attributes*, each of which is a pair consisting of a *type* and *value*. Types and values are typically strings, though some values may also be binary packed representations. We encourage and expect applications to expose as much *metadata* as possible about an item, including application data. Two examples are shown in Figure 2(a), representing a message from Bob to Alice, and a photo of sunset. Note that we do not require users to enter more metadata about their objects than applications would require themselves; the value of exposing metadata is in the ability to search and organize data.

Links between Data Objects: Data objects can be linked into a directed graph to either represent prerequisite dependencies or ownership information. For example, a photo album's metadata can link to the set of photos in the album, a webpage can link to its embedded objects, or an email can link to its attachments. This explicitly exposes the structured relationship between data objects more richly than directory hierarchies. Applications can also express an "ownership claim" over objects by linking its application object to the desired data objects. For example, a web browser may lay claim over cached objects, and an email reader may claim stored emails.

Since Haggle allows many applications to claim objects, it does not have a "delete" call. Instead, Haggle implements lazy garbage collection, to allow searching for unreferenced objects and delay space reclamation until it is needed.

Object Filters: To facilitate searching, the data manager supports searching for objects using a filter object which comprises of a set of regular-expression-like queries over the attributes. For example, a query might include: $(mimetype = text/html \wedge news \in keywords \wedge timestamp >= (yesterday))$ Filters can be one-time searches, or made persistent to "watch" for new or incoming matches. Filters can also be made local or remote, effectively providing a "subscription" mechanism.

4.3 Scheduling and Managing Just-in-time Resources

Network interfaces are shared mediums which consume device and user resources, in terms of time, energy, and cost. To manage and schedule multiple network interfaces, requests for network use from components and applications are delegated to the resource manager. The resource manager considers the set of outstanding tasks and determines which tasks are allowed to execute by evaluating whether it is beneficial and cost effective when taking into account the user's preferences and policies. The centralized resource management design enables Haggle to schedule network resources in a manner coherent with the user's policies and behave in a manner understandable by the end user.

Because certain operations are time or sequence critical, there are two types of tasks supported by Haggle: asynchronous and immediate. Asynchronous tasks can be delayed or scheduled by the resource manager at any arbitrary time. Immediate tasks are evaluated right away and a decision for whether or not to execute a task is based only on the current context.

Due to the dynamic scheduling of tasks and potentially changing mobile environment, the benefits and costs of asynchronous tasks can also vary over time. For example, an email checking task is less beneficial if email was last checked 1 minute ago, but more beneficial if over an hour has elapsed. Similarly, as the connectivity environment changes, the costs for operations can dynamically change.

Once a task is being executed, the resource manager can also be asked for an extension on the resource use if the scope of the work being done by the task needs to be increased beyond the initial cost/benefits specified. This is useful for circumstances such as email checking, where we may find a large attachment waiting for download.

The task model is in marked contrast to the traditional network stack, where networking operations proposed by applications or operating system functions are always attempted. The centralization of decision-making about what tasks are worth doing at all, and which are more important at any time, allows Haggle to have a number of advantageous features. First, Haggle can easily and intuitively manage the use of multiple connectivity interfaces. Haggle's support for late-binding protocols and names allows it to manage which subset of connectivity interfaces to use and what kinds of tasks are allowed on those interfaces. Second, Haggle can easily enable dynamic scheduling and prioritizing of tasks. For example, instead of checking email at fixed intervals, the checks can be more often when bandwidth and energy are abundant, and less often otherwise or when there are more important tasks. Similarly, applications are free to request operations of varying priorities, including speculative operations, which are often not possible or worthwhile, but automatically executed when the right opportunity arises.

The current prototype implementation only considers costs in terms of time-on-network, which provides an estimate of energy consumption. We do not yet support costs in terms of monetary charges per byte or quota limits. Ideally the network interface card or driver would provide power consumption estimates since they have a greater knowledge of their radio characteristics and medium state. We currently assume all nodes are cooperative, and are not using policies which limit interactions with peers.

5 Support for Existing Applications

Based on our introductory motivating examples, we have chosen to target email and web as our prototype applications. To be clear, by "email" and "web" we mean the applications, rather than the protocols that underlie them.

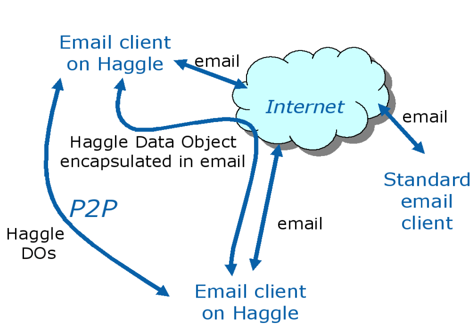
Both of these applications enjoy significant support from the pre-existing infrastructure deployment of servers and content. It is a crucial feature of Haggle that we can take advantage of this infrastructure as well as providing new functionality. This makes Haggle much more compelling to existing users of that infrastructure, and the value added by Haggle provides motivation for its deployment.

To provide legacy support for existing email and web applications, we implement localhost SMTP/POP and HTTP proxies as Haggle-native applications. This allows users to keep using the same applications they habitually use (we have tested Outlook Express, Thunderbird, Internet Explorer and Firefox) with only minimal reconfiguration. We will first describe how Haggle provides support for email, followed by the description of web support.

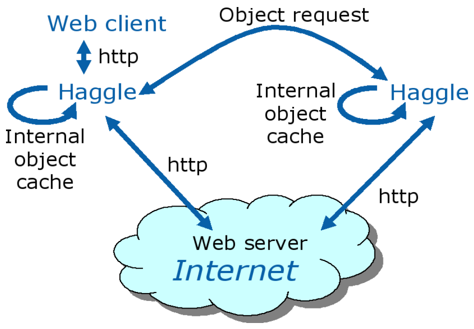
5.1 Email

Supporting email in Haggle consists of two components: an SMTP/POP proxy for interfacing with email clients, and SMTP and POP Protocols inside Haggle that communicate with email servers.

The SMTP proxy accepts emails provided by the user's email client and translates them into linked data objects using Haggle's API. The proxy uses the recipient field of the email header to search for an appropriate *name* which describes the intended recipient, as illustrated in Figure 2(b). The proxy then creates a forwarding request to send the mail object to the individual described by the name object. Haggle now will



(a) Email Application



(b) Web Application

Fig. 3: Haggle Email and Web Applications

dynamically decide when the message will get delivered, the protocol to use, and over which network interface.

Similarly, when the user's email client checks for new mail, the POP proxy uses the data manager to search for newly arrived messages. New messages are reconstructed as email messages (including attachments) and returned to the email client.

When infrastructure connectivity to the Internet is available and the recipient is not nearby, Haggle will use the existing email infrastructure to deliver the email message. This is possible when the 802.11 connectivity reports that it has access to the Internet. The direct forwarding algorithm and SMTP protocol plugin will both report their ability to resolve the name graph to a deliverable end-point. The resource manager will then determine if there is sufficient benefit to execute the delivery. If so, the direct forwarding algorithm will use the email protocol to transform the message object and use the SMTP protocol to deliver the email using 802.11 connectivity.

If Haggle decides to use a peer-to-peer connection, whether due to lack of infrastructure availability or to improve throughput, the two peers rendezvous to form an ad hoc network. The sending node then establishes a peer-to-peer connection to the receiver, and sends the message as a Haggle object, complete with all necessary links and meta-data.

5.2 Web

The Haggle web proxy operates as a normal web proxy when Internet connectivity is available. As requests are retrieved and returned to the browser, the web proxy stores the information in the Data Manager, including link and object relationships. The mechanism for resolving web addresses and retrieving web objects is similar to the description for the email application, except instead of email protocols, the web plugin is able to understand URL addresses as names and communicate HTTP protocols. If Internet connectivity is not available or too expansive to use, then the proxy creates a filter subscribing to the URL. A notice page is returned to the browser notifying the user that Haggle is attempting to service the request. This page refreshes itself occasionally so that the webpage will be displayed automatically when it arrives.

If a peer has matching cached content, the requested URL and linked embedded objects are sent back to the requester. If a peer is willing or has an incentive [3] to bridge the request to the Internet, the HTTP protocol first downloads the requested URL, parses it to look for embedded content, and downloads the necessary objects. The linked object is then sent back to the requester.

6 Experiments and Results

In this section we describe several experiments using the motivating applications described earlier. We provide qualitative results showing the new capabilities that Haggle enables, in addition to quantitatively demonstrating that Haggle's implementation, although not optimized, has acceptable overheads.

The Haggle implementation has been developed using Java J2ME CDC, which means it is compatible with PC and notebook platforms (e.g. Windows, Linux) as well as mobile platforms (e.g. Windows Mobile, Nokia tablets running Linux). This development has been conducted using sourceforge.net, under the GNU General Public License (GPL), available at http://sourceforge.net/projects/haggle.

We conducted the experiments on two laptop computers, which we will call *node1* and *node2*. Both are running Windows XP. Node1 is equipped with an Intel 3945 mini-PCI 802.11 interface, and node2 is equipped with an Intel 2200 802.11 interface. For infrastructure connectivity, the nodes connect via wireless 802.11g to an access point with access to the Internet.

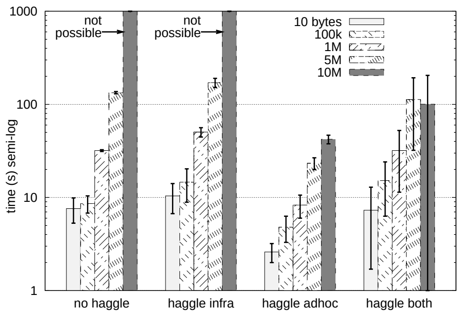
Email: For the email experiments we created several accounts using the Google Mail (GMail) service. GMail provides POP and SMTP services over an encrypted and authenticated SSL link. This allows us to have one configuration which works from within any network that allows Internet access. However, there are limitations with using the GMail service. Though there is no limit for the size of email received, GMail restricts the size of outbound emails to be 10 megabytes or less.[5]

For the quantitative experiments we send emails of varying sizes, ranging from 10 bytes (no attachment) up to a 10 megabyte attachment, from node1 to node2. For each size increment seven unique emails are sent over a 3 Megabit download / 800 Kilobit upload DSL link. An automated script is used to send an email from node1 to node2, with node2 configured to check its inbox once every 5 seconds. The script ensures that for every email that node1 sends, node2 must receive it first before node1 sends the next email. This eliminates any congestion effects in the results.

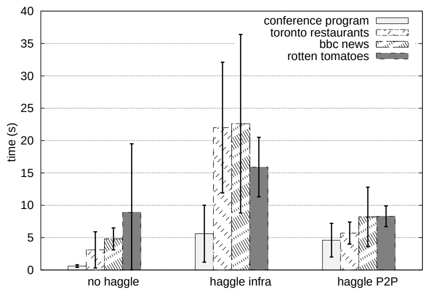
Figure 4(a) shows the latencies for end-to-end delivery of various-sized emails under different network connectivity conditions, both with and without Haggle. The *no haggle* and *haggle infra* clusters provide a comparative baseline between email clients as normal versus using Haggle forced to use infrastructure connectivity, respectively. The results show that Haggle imposes a low overhead.

The most important result is shown in the *haggle adhoc* cluster, which shows Haggle sending and receiving emails without infrastructure present. This operation is not

[5] limit raised to 20 megabytes at time of publication.



(a) Mean email end-to-end delivery times. Individual bars indicate attachment size.



(b) Mean webpage retrieval times.

Fig. 4: Email and web experiment performance. Both graphs show standard deviations as error bars. Lower values are better. Note that in 4(a) for both *no haggle* and *haggle infra* cases it was not possible to send 10M emails due to server limitations. 4(b) does not show *no haggle P2P* because it is not possible to access web pages using existing software.

possible using the email client alone and would have corresponding graphs of infinite height. The ad hoc transmission of emails, shown by the *haggle adhoc* bars, is fastest since it uses a direct transmission in ad hoc mode. The other modes of operation require use of the access point link, which includes going out the DSL line, through the Internet for both transmitting the email as well as retrieving. We note that having ad hoc transmission ability can also overcome limitations of infrastructure based services, as seen in the 10Mb attachment test. Gmail places a size limit on the mail size, which prevents large emails from being sent.

The *haggle both* bar shows Haggle performing in an environment that has infrastructure access but Haggle has the option to communicate in ad hoc mode when appropriate. Ideally, the *haggle both* performance would be close to the *haggle adhoc* performance. This is not so (though it is still comparable with *no haggle*) and there is a larger variance in the numbers. This is due to interaction between the 802.11 connectivity and the POP protocol. Because an Internet neighbour is visible in this scenario, the POP protocol is requesting tasks to check the email account. This is additional work that Haggle is not doing in the *haggle adhoc* case. Added to this is the significant overhead incurred by 802.11 in switching between ad hoc and AP modes due to lost DHCP request packets. This overhead is not inherent to Haggle, but rather a limitation in the current implementations of 802.11 which can be overcome using techniques such as MultiNet [6].

Web: In our web experiments we focus on the retrieval of static and dynamic pages from content providers. We chose four different webpages to cover a range of complexities, sizes, and scenarios. All of these sites represent classes of content which users would like to look up and, in the right mobile context, have a reasonable expectation that other users around might have similarly matching content.

– Conference Program: This page is for a conference's technical program, which is relatively simple consisting mostly of text and no dynamic content, with a transfer size of 64Kb. Attendees at the conference are likely to request this page frequently to see what is on next; however, wireless networks at conferences can frequently encounter connectivity failures [11].
– City Life is a popular city life and culture site with moderately complex layout. The transfer size is 500Kb, sent as 380K of gzip-enabled web traffic.
– BBC news is a relatively complex website with frequently updated content. The transfer size is 370Kb, sent as 100Kb of gzip-enabled web traffic. This page is highly viewed, so there is a reasonable likelihood of a copy being present in a group of users.
– Rotten Tomatoes is a movie review site which contains a dense layout with dynamic content. The transfer size is 834Kb, sent as 230Kb of gzip-enabled web traffic. This might be looked for at a cinema while deciding what to watch, with a reasonable expectation that others in the area already looked it up.

For each of the experiments we retrieve the contents seven times, each time clearing all caches. We measure the end-to-end time as starting from the moment of request at the browser until the browser finishes loading and rendering all content on the page, using the Firefox web browser with the FasterFox plugin since it contains a built-in page load timer (we turned off all other functionality that FasterFox offers).

Figure 4(b) shows the performance results for retrieving the above described webpages with and without Haggle. Between *no haggle* and *haggle infra* the comparison is less favourable than for the email case. We attribute this to the overheads of (a) the time taken to parsing the HTML pages to determine linked data objects, (b) the overhead in the proxy approach, since the web client opens and closes multiple socket connections to inform Haggle of different objects it requires, (c) inefficiencies in our Data Manager implementation in that the webpages are stored data persistently before they are transmitted to the web client.

For each web object retrieved from the Internet, the web proxy attempts to reconstruct its relation with other objects it was linked from. Because web browsers make multiple simultaneous connections to the proxy and use each pipe in parallel, we must examine the headers of the objects returned in order to properly associate objects to webpages. To do this the web proxy examines the referrer tag of the HTTP response message for the retrieved object to determine from which other object the current was requested from. After finding the originating object, the web protocol creates a link from it to the newly retrieved web object. This search and link requires queries to the data manager in Haggle which adds overhead time to each web object retrieved, visible in the comparison between *no haggle* and *haggle infra*.

For the *haggle P2P* experiments, we have node1 configured to enable access to the access point as well as communicate in ad hoc mode. The four webpages described are then visited using the web client on node1 so that it has a cache of data objects representing those pages and embedded objects. At this time, the access point is turned off, modelling node1 being moved to an infrastructure-free location. Node2 is only able to communicate in ad hoc mode, and is placed near node1. We request a webpage on node2 (clearing the cache each time an experiment is run); since node2 does not have

Internet connectivity, it sends a filter requesting the webpage to node1, who returns the matching webpage with embedded objects. This last experiment shows fundamentally new functionality enabled by Haggle for the web browser, in that it can now operate even when there is no infrastructure Internet connectivity..

7 Discussion and Future Work

Haggle's architecture enables new applications to be created easily, taking advantage of the flexibility that Haggle provides. One interesting application that we are targeting for future work is in the area of "resource-friendly media sharing". We observe that humans collect ever more media (photos, music, videos, etc) and wish to (a) share them easily with friends, and (b) have them transferred seamlessly between their devices, both mobile devices and those fixed at various locations.

In current networks, it is not possible for an application to easily express "all photos taken on my mobile phone should be sent to my home server for backup" without being at risk of consequences such as large GPRS bills when their phone transmits holiday snaps over a foreign carrier, and the phone running out of batteries since it performs transfers even if there is scarce power. With Haggle, these concerns can be easily expressed, and persistent remote filters provide a simple yet powerful publish-subscribe mechanism for this kind of application.

Another feature easily enabled using Haggle is the predictive and prefetching download of content. For example, web clients can ask for low-priority predictive downloads of webpages that users might need because they are often-viewed or linked form the currently viewed page. Such predictive requests are easily expressed in Haggle using lower application benefit levels. Haggle is able to automatically allow or disallow these predictive operations based on the user's policies of energy and cost constraints.

A further interesting avenue of research is to investigate how users will perceive the networking world with Haggle. We observe that users currently have a simple mental model of mobile networking. When they have an IP connection, their apps work, otherwise they don't. Haggle breaks this model for the good, as it provides more functionality. How will users mentally model this flexibility? How will they understand what works under Haggle and what fails? One possible way in which users can be trained is to consider if they can see what they need in the environment. If a user can see the person that they are messaging, or they can see others who have data that they want, then they should expect that Haggle might deliver that message or find that data.

With the growing popularity of web service applications such as web-based office suites, many applications are being re-developed on the client side to take advantage of these services. However, despite the growing availability of affordable broadband-speed cellular services, request latency [5] requires clients to use smart local caching and prefetching strategies to give users a smooth experience. Haggle provides a simple networking model for creating these client-side applications while simultaneously enabling significant support for utilizing peer-provided resources.

7.1 Future Work for the Haggle Prototype

The currently experimental prototype for Haggle matches for web objects based on exact URL addresses. As motivated by our example, users seeking information often use search functionality and are happy to receive results from any number of different websites. Because Haggle already supports regular expression matching of attribute contents, we expect adding basic searching capability for neighbourhood cached content to be a powerful new feature. In addition, for nodes willing to bridge search queries to the Internet, we plan to add protocols which interface with search engines to perform queries and retrieve the first few query results.

We also plan to extend the prototype to support additional interfaces such as Bluetooth, cellular, and even ultra-wideband connectivity. Many other research projects have explored the problems of heterogeneous network interfaces [21], and struggled with how this can be done sensibly using IP routing. Haggle offers a new approach to this problem, complementing existing IP-based approaches [25].

A limitation of the current Resource Manager is that it is reactive only, and does not attempt to predict future connectivity options (e.g. as OCMP [19]). For example, currently Haggle may epidemically send a message to a remote host when, in five minutes, the user will arrive at their place of work and have free broadband connectivity. A related feature is to enable streaming support by adding reservation capabilities in cooperation with connectivity predictions to provide smoother experiences.

We do not use monetary costs or energy consumption in our current decision process, however these are key issues in device connectivity today, as they impact battery life and the potentially high cost staying "always-on". Particularly when we have multiple connectivity interfaces, we will likely be faced with situations where we have to choose between connectivity options which trade off forwarding benefit against monetary cost or energy consumption.

7.2 Security and Privacy

In the current version of Haggle, security and privacy have not been addressed as key concerns in order to narrow the scope of the problem. We intend to introduce security primitives as a core concern in future versions of Haggle. In the following discussion we have made an initial analysis of the potential security threats that Haggle raises.

Many data security issues in Haggle can be handled using standard security techniques such as encryption, access control, and data signing. Haggle merely makes it more feasible to launch a man-in-the-middle attack. One proviso is that many security techniques rely on access to a trusted third party, e.g. a certificate signing authority. This access may be available less often when using Haggle. One interesting approach would be to accept data which is signed but unverified and taint it as "untrusted" (both internally and to the user) until the signature can be checked and verified.

There are particular security and privacy issues in the use of name graphs. A name graph can contain sensitive information, e.g. a user's email address and/or phone number, or the number and type of a user's devices. A possible solution is to restrict trust of certain names to particular groups of users, such as circle of friends or known personnel of a company. Tackling this problem is left for future work.

8 Related Work

Many previous efforts have individually addressed late-binding interfaces or names to provide flexibility across dynamic environments. This work extends previous efforts [18] in providing a novel node architecture for applications by providing a clear resource delegation model for late binding interfaces, protocols, and names.

Late-binding interfaces allow devices to make better use of their available connectivities, utilizing their strengths and minimizing the impacts of their weaknesses. Horde [17] presents a middleware system which can stripe across different wireless radios according to user specified profiles. Wang [25] presents a policy based hand-off system which allows users to specify the best wireless communication system to use.

Late-binding name systems allow applications and services to rendezvous based on descriptive names over a self-organizing overlay network. Decoupling naming from the physical addressing provides a clean abstraction for supporting dynamic and mobile nodes in the network topology as well as routing based on new metrics such as location and domain specific contexts. The i3 [22] system hashes the name identifier space into a DHT overlay network, allowing applications and services to rendezvous at the same overlay node, independent of node mobility. INS [1] allows applications to specify names as trees of key-value pairs expressing the desired service or device. Each node participating in the routing overlay network can perform matching functions to determine where best to forward the request in order to find a matching destination.

Other projects such as OCMP [19] have similar goals in providing a node architecture for supporting applications. However these efforts are mostly focused on routing, particularly for challenged environments [8]. Haggle provides a more general node architecture for the provisioning, scheduling, and late-biding of network resources independent of applications.

9 Conclusions

Haggle is a new node architecture for mobile devices that enables seamless network connectivity and application functionality in mobile environments. By separating the networking concerns from the application, Haggle enables delegating network operations to a central resource manager on the device which can effectively select the right just-in-time bindings for network interfaces, protocols, and names in accordance with user-defined policies. We demonstrate the effectiveness of Haggle's approach using existing email and web applications on a Haggle prototype. Our experiments showcase the ability to dynamically select the best network operating mode when transferring emails and function even when disconnected from infrastructure. This allows people to use the same application across different connectivity scenarios, something that today would at best require manual configuration, and at worst be impossible.

References

1. W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of SOSP 1999*.

2. M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Proceedings of PODC '99*.
3. G. Ananthanarayanan, V. Padmanabhan, C. Thekkath, and L. Ravindranath. Collaborative downloading for multi-homed wireless devices. In *HotMobile 2007*.
4. H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for internet hosts. *SIGCOMM Comput. Commun. Rev.*, 29(4):175–187, 1999.
5. R. Chakravorty, A. Clark, and I. Pratt. Gprsweb: optimizing the web for gprs links. In *Proceedings of MobiSys*. ACM Press, 2003.
6. R. Chandra, P. Bahl, and P. Bahl. Multinet: Connecting to multiple ieee 802.11 networks using a single wireless card. In *Proceedings of IEEE Infocomm 2004*.
7. E. Cutrell, S. T. Dumais, and J. Teevan. Searching to eliminate personal information management. *Commun. ACM*, 49(1), 2006.
8. K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of SIGCOMM 2003*.
9. B. Ford, J. Strauss, C. Lesniewski-Laas, S. Rhea, F. Kaashoek, and R. Morris. Persistent personal names for globally connected mobile devices. In *Proceedings of OSDI*, 2006.
10. P. Hui, A. Chaintreau, J. Scott, R. Gass, J. Crowcroft, and C. Diot. Pocket Switched Networks and human mobility in conference environments. In *Proceedings of WDTN 2005*.
11. A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding congestion in ieee 802.11b wireless networksrevised. In *Proceedings of IMC 2005*.
12. M. Karsten, S. Keshav, and S. Prasad. An axiomatic basis for communication. In *Proceedings of HotNets 2006*.
13. J. Leguay, T. Friedman, and V. Conan. Dtn routing in a mobility pattern space. In *Proceedings of WDTN 2005*. ACM Press.
14. A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. In *Proc. SAPIR*, 2004.
15. M. Mauve, A. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *Network*, 15(6), Nov 2001.
16. C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector routing. *RFC3561*, 2003.
17. A. Qureshi and J. Guttag. Horde: separating network striping policy from mechanism. In *Proceedings of MobiSys 2005*. ACM Press, 2005.
18. J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: a networking architecture designed around mobile users. In *Proceedings of IFIP WONS 2006*.
19. A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *Proceedings of MobiCom 2006*.
20. R. C. Shah, S. Roy, S. Jain, and W. Brunette. Datamules: Modelling a three tiered architecture for sparse sensor networks. In *IEEE SNPA 2003*.
21. J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: hierarchical power management for mobile devices. In *Proceedings of MobiSys 2005*.
22. I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of SIGCOMM 2002*.
23. J. Su, J. Scott, P. Hui, E. Upton, M. H. Lim, C. Diot, J. Crowcroft, A. Goel, and E. de Lara. Haggle: Clean-slate networking for mobile devices. Technical report, University of Cambridge, 2007. UCAM-CL-TR-680.
24. A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical report, Duke University, 2000. CS-200006.
25. H. J. Wang. Policy-enabled handoffs across heterogeneous wireless networks. Technical Report CSD-98-1027, 23, 1998.
26. W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of MobiCom 2004*.