Haggle: a Networking Architecture Designed Around Mobile Users

(Invited Paper)

James Scott*, Pan Hui†, Jon Crowcroft†, Christophe Diot‡
*Intel Research Cambridge
james.w.scott@intel.com
†Cambridge University
firstname.lastname@cl.cam.ac.uk
‡Thomson Research
christophe.diot@thomson.net

***Abstract*— Current mobile computing applications are infrastructure-centric, due to the IP-based API that these applications are written around. This causes many frustrations for end users, whose needs might be easily met with local connectivity resources but whose applications do not support this (e.g. emailing someone sitting next to you when there is no wireless access point). We identify the general scenario faced by the user of Pocket Switched Networking (PSN), and discuss why the IP-based status quo does not cope well in this environment. We present a set of architectural principles for PSN, and the high-level design of Haggle, our asynchronous, data-centric network architecture which addresses this environment by "raising" the API so that applications can provide the network with application-layer data units (ADUs) with high-level metadata concerning ADU identification, security and delivery to user-named endpoints.**

I. Introduction

End user experiences of mobile, many-device computing are often marked by frustration and inconvenience. Users are forced to be highly aware of their connectivity environment, with many applications only working when networking infrastructure is available. One ubiquitous example is that of two people with laptops sitting next to each other, who cannot email a file they wish to share because infrastructure is either unavailable, not working properly, or too costly. While there are other ways to send the file, this requires training and further understanding of the network situation – most often, people simply fall back on the use of USB key flash drives. The billion US dollar market for these in 2005 [1] is a testament to the failure of the mobile networking research community to provide a network architecture that supports truly mobile applications.

In this paper, we make the following contributions aimed towards beginning to address that failure. We first provide a formulation of the networking environment faced by mobile users, a scenario which we term Pocket Switched Networking (PSN) (Sect. II), and describe how the status quo of TCP/IP is unable to cope with PSN (Sect. III). We present a set of architectural principles which we believe will enable a network architecture to perform well under PSN conditions (Sect. IV). We then describe Haggle, our clean-slate design for a new network architecture following these principles (Sect. V). Finally, we present related work (Sect. VI) and conclusions (Sect. VII).

II. Pocket Switched Networking

In designing a new network architecture, it is first important to define the scenario in which that architecture will be used. IP, for example, was designed against a backdrop of a multitude of existing networks, and with the primary needs being resilient end-to-end communications in the presence of node failures, as befits its originator, the US Department of Defense [2].

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

Thus, in PSN, there are three methods by which data can be transferred, namely neighborhood connectivity to other local devices, infrastructure connectivity to the global Internet, and user mobility which can physically carry data from place to place. For the former two methods, the connectivity is subject to a number of characteristics, including those of bandwidth, latency, congestion, synchronicity (e.g. email or SMS are asynchronous, while ad-hoc 802.11 is synchronous), the duration of the transfer opportunity (i.e. the time till the device moves out of range), and also monetary cost (usually only for infrastructure). For the latter method of user mobility, users acting as "data mules" can transfer significant amounts of data, and while users' movements cannot in general be controlled, they can be measured [3], and patterns in those movements can be exploited.

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

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

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

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

III. Problems with status quo

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

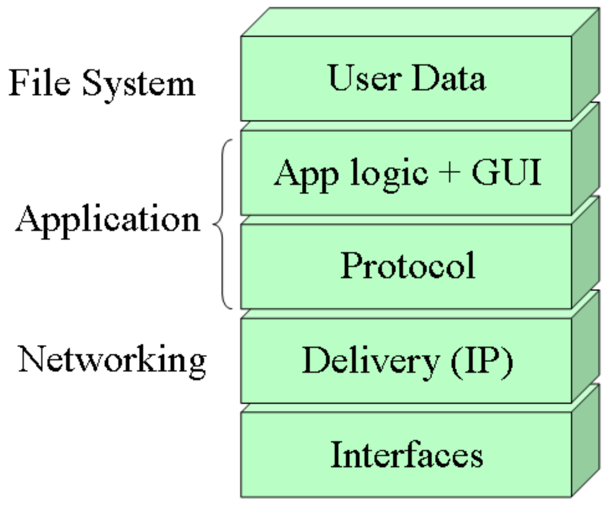

Fig. 1. Current networking architecture for mobile applications

The root cause of this is the fact that applications are provided with a networking interface that only understands streams of data directed at anonymous numeric endpoints (namely TCP/IP). As illustrated in Figure 1, this forces developers to implement protocols for naming, addressing and data formatting internally in the applications themselves, e.g. SMTP, IMAP and HTTP. While at the GUI level, applications have general user-level tasks such as "send this file to James," once a particular network protocol such as SMTP is imposed on that task, it becomes the a more specific task, e.g. "send this file to the server pointed at by the MX record in the DNS record of the domain name part of james.w.scott@intel.com". The latter task is specific to a particular kind of connectivity scenario, in this case infrastructure-based. It is therefore impossible to execute even if James's device is in the neighbourhood at that time — i.e. even if the user-level task could be satisfied.

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

A third problem is that persistent user data is kept by applications in a file system which, in the current node architecture, is unconnected from the networking system (again illustrated in Figure 1). This means that all "sharing" of data between nodes must often be conducted by applications themselves[1]. The biggest example of this is the device synchronisation problem — when a user has multiple devices, they must explicitly run an application on each which pulls their data out of the file system and shares it with their other device(s). Such synchronisation is often a source of much inconvenience for users, since the sync tools must understand the different ways that each user application uses the file system to store data and metadata, and often has to translate it so that different applications can be sync'd with the same data. Another example is in distributed web caching — the exact web page that a user wants may be in the cache of a neighbouring node, but since web browsers do not explicitly support the transfer, there is no way to get this off the neighbour's file system and into the network to be shared with the user.

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

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

IV. A New Set of Mobile Networking Principles

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

A. Forward using application layer information

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

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

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

B. Asynchronous operation

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

C. Empower intermediate nodes

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

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

D. Message switching

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

E. All user data kept network-visible

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

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

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

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

F. Build request-response into the network

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

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

G. Exploit all data transfer methods

The aim of Haggle is to take advantage of all the communication opportunities offered by the PSN environment, including local connectivity with neighbouring nodes, and global connectivity using infrastructure. Human mobility patterns can be exploited by using forwarding algorithms which target nodes known to have mobility patterns which are likely to be useful, e.g. because they have seen a destination node recently [4], or because they share the same mobility pattern as a destination [5].

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

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

H. Take advantage of brief connection opportunities

In the PSN scenario, connection opportunities can be fleeting, e.g. when walking or driving past another mobile user or an AP. It is therefore important for Haggle to be able to take full advantage of time-limited connection opportunities, by prioritising potentially exchanged data so that the most urgent data is sent first, and by using underlying protocols which make efficient use of bandwidth during short connection opportunities [6]. This also implies that neighbourhood discovery (neighbours meaning both mobile devices and APs in this case) is a key part of Haggle, since transfer opportunities must be detected in a timely fashion.

I. Empowered and informed resource management

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

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

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

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

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

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

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

J. Use and integrate with existing application infrastructure where possible

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

V. The Haggle Architecture

Haggle is the name given to our new network architecture, which applies the principles outlined above to overcome the problems with the status quo and effectively operate in the PSN environment. Haggle is an unlayered architecture which internally comprises four modules; delivery, user data, protocols and resource management, as shown in Figure 2. As compared to the current network architecture in Figure 1, we immediately notice a number of high-level differences:

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

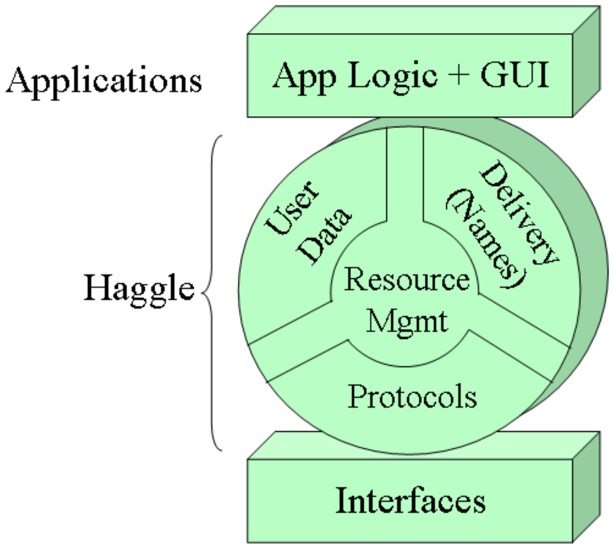

Fig. 2. Overview of Haggle architecture

We now discuss the functionality of the four modules in more detail.

A. User Data

Application Data Units (ADUs) are Haggle's format for user data, and as the name suggests they are an encapsulation for a data item meaningful to an application such as a photo, a music file, a webpage, a message, etc. In the spirit of Application Layer Framing [7], ADUs are not only the unit of data for applications, but are also the unit used for transfer between Haggle nodes, i.e. they are the messages in message switching. An ADU is comprised of many attributes, where an attribute is a type-value pair. The type is always a text string, the value may also be a text string, but may also be a binary stream, e.g. for the mp3 file comprising the main content of an ADU representing a song. The intent is for rich metadata about each object to be exposed as attributes — such data could be duplicated in the binary stream section if this is useful for the application.

An ADU attribute can be used to store such information as:

- Descriptive information for user data, e.g. keywords for a picture.
- Document management information, e.g. the creation-date, creation-user, modification-date, etc.

- Security and permissions information, e.g. the list of local users with access permission, whether the item must be encrypted on leaving the node, and information to the encryption method and key

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

Thus, ADUs can also contain information about forwarding state such as:

- a list of destination names and addressing hints for those names (there can be more than one destination, each with more than one way to get there).
- details of the source node.
- a list of forwarding hints, informing intermediate nodes of dynamic information (e.g. X-was-seen-at-time-T-by-Y)
- a list of nodes which it has passed through
- timeout/flooding avoidance parameters — e.g. max duplication count, max hop count, deadline
- a priority level specified by the source node
- security information, e.g. an authentication signature or encryption details for the claimed ADUs
- any other data relevant for forwarding tasks — since the ADU format allows a variable number of attributes, a new forwarding algorithm can easily send other data.

1) ADU Example: The example ADU below represents a message including a photo, some text, and forwarding state for the message.

Filename	DSC10027.jpg
Mime-Type	image/jpeg
Creation-date	1/1/2006 17:32
Created-by	James Scott
Security-group	Public
Keywords	Athens, Greece, seashore, sunset
Data	[binary jpeg data]
Mime-type	text/plain
Date	1/1/2006 17:40
Data	”Wish you were here!”
Date	1/1/2006 17:40
From	“James Scott”; james.w.scott@intel.com
To	“Jon Crowcroft”; +447123456789; jon.crowcroft@cl.cam.ac.uk
To	“Pan Hui”; pan.hui@cl.cam.ac.uk; BT 0F:CC:3E:C9:87:21
Priority	5
TTL-hops	100
TTL-deadline	2/1/2006 17:40

Note that this ADU comprises three sections, similar to emails with different message parts. Thus, the photo could be easily split off from the rest of the message, e.g. by an intermediate node whose user had specified an interest in photographs of sunsets, while the whole message was delivered. In implementation, the photo itself would not be duplicated on a node’s disk, but instead pointers to the same data file would be used for efficiency.

B. Protocols and Naming

A key feature of Haggle is the use of user-level naming schemes for data transfer decisions. This immediately raises the question of how these high-level names get translated into lower-level addresses that the physical network interfaces can use for transmission. In other words, what is the equivalent of ARP for Haggle?

In order to answer this question, we must examine what an “address” is for Haggle. We define an address as any name for which there is a protocol available in the Protocols module (see Figure 2) which is capable of sending the ADU to that address. Different connectivity situations and different applications require different address types, for example, the use of local connectivity to share ADUs with a neighbour might use a Bluetooth or 802.11 MAC address as a Haggle address, while when using infrastructure, an email address can be used as a Haggle address (if an email protocol is supported). Note that different nodes might regard different types of name as “addressable” because of their different capabilities — to a node with geographic information and forwarding capabilities, a GPS coordinate is an addressable name, while to another node that name needs to be mapped onto another name type before forwarding can occur. Mechanisms for mapping between names are therefore very important in Haggle.

There are many methods by which names can be mapped to other names. An ADU can contain mappings between names, e.g. the ADU example given above, which specified mappings between a non-addressable personal name, and email, telephone and Bluetooth MAC addresses which various underlying protocols could use. Names can also be dynamic, e.g. the current IP address of an infrastructure-connected mobile node, which changes from time to time, or the current location of a node (if a geographic routing protocol is available). Both static and dynamic name mappings may be found in the ADU itself, or may be found other ADUs acting as name lookup tables on the local node. Such naming ADUs might be created by applications (e.g. contact details for a particular person) or by forwarding algorithms (e.g. using neighbourhood device discovery). The use of a standard ADU format for naming has the benefit of allowing different nodes, applications, and forwarding module implementations to parse naming ADUs from each other. The flexible nature of ADUs also means that different naming schemes can be constructed easily, enabling the use of Haggle for new applications and protocols in a way that highly-specified lookup tables (e.g. DNS MX records) do not.

C. Neighbours and Forwarding

As previously mentioned, Haggle must perform neighbour discovery in order to take advantage of connection opportunities. The result of neighbour discovery is that some set of addresses are marked as "nearby". One example might be Bluetooth inquiry, which would result in a set of Bluetooth MAC addresses being marked as "nearby", while another example would be that when an accessable AP is seen, all Internet domain names would be considered "nearby". Forwarding algorithms in Haggle estimate the "benefit" of performing a transfer of a given ADU (or set of linked ADUs) to a given nearby node. While some transfers are obviously beneficial, e.g. the transfer of an ADU to an email address which it lists as a destination, other transfers are less obvious, e.g. the transfer of the same ADU to a node which is not the final recipient but might be willing to help in the transfer process, or the transfer of a ADU requesting content to a particular neighbour who may or may not help provide the content requested.

D. Resource Management

All use of resources in Haggle is controlled by the resource management module. This operates by performing a cost/benefit analysis on "tasks" that other modules specify. The forwarding module, for example, specifies a number of potential transfers as "tasks" with associated benefit estimates, and also gives enough information so that the "cost" of those tasks can be estimated in terms of resources consumed, including the use of network bandwidth, battery power, monetary cost, etc.

The resource management module compares the cost with the benefits to decide what action to take next. In addition, the resource management module can use context-based information to estimate the *scarcity* of resources, e.g. the network bandwidth available, the expected time until battery charging can take place, etc. This can be incorporated into the cost-benefit decision by raising or lowering the value placed on certain resources dynamically.

By performing resource management centrally, we allow applications to cooperate in sharing resources rather than competing, since applications can specify priority levels for various actions and allow low-priority actions to avoid using scarce resources. We can also provide the user with the ability to specify global preferences for the device, e.g. regulating the consumption of expensive bandwidth to an acceptable level.

E. Interacting with applications

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

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

VI. Related work

In this section we discuss related work by ourselves and by others.

As with many pieces of research, the proposed architecture above creates as many questions as it answers. There are many challenges faced in PSN [8]. These include the problems of designing forwarding algorithms for the PSN environment, of creating suitable naming schemes and mapping those names onto deliverable addresses, of security and privacy protection, and of usability when there are no end-to-end guarantees.

Our approach towards these challenges is practical rather than theoretical, using implementation, deployment, and measurement. The architecture design above is being implemented for mobile platforms such as mobile phones and PDAs, and will be tested with real applications and real users. This will allow us to hone the architecture to address real-life situations, and, in collaboration with others, to address the various challenges detailed above.

Human mobility for data transfer has been explored by a number of different research groups, including under the names of "data mules" [9] and "message ferries" [10]. In Haggle, our approach has been to perform measurements of human mobility patterns [3], [11]. These have found that human mobility has significantly different characteristics to those assumed in simulations which have previously been used to evaluate neighbourhood forwarding algorithms, e.g. in mobile ad-hoc networks which have relatively dense networks. Such measurement traces can be used to help design and evaluate forwarding algorithms for the PSN environment.

Delay-Tolerant Networks (DTN) [12], [13] focuses on protocols addressing scenarios where TCP/IP networking is not feasible, with two such scenarios being when there are large time delays (e.g. in interplanetary networks), or when there is no contemporaneous end-to-end link, e.g. when using a "message ferry" to physically carry data to remote locations. Haggle shares some of its principles with DTN, such as the use of message switching and opportunity-oriented networking, but is additionally exploring ideas such as the mapping of user-level names onto many parallel delivery methods, the exposure of all user data to the network, the use of request and response as network primitives, and the key role of resource management.

The data-centric networking aspect of Haggle is similar in nature to a number of efforts, including FreeNet [14] and Distributed Hash Tables (DHTs) such as Chord [15] in which a hash of an object is used to locate an object, and peer-to-peer networks such as eMule which allow text searches over metadata such as the file name to find objects. In Haggle, we aim to perform data-centric networking in the PSN environment, which does not have the relatively stable connectivity assumed by the previous work.

VII. Conclusions and Future Work

Haggle is a network architecture designed from the ground up around the needs of mobile users as characterised by the Pocket Switched Networking environment. We have described the motivation for a new architecture, and the principles behind Haggle's design. We plan to build an open source, cross-platform implementation of Haggle for mobile devices, and trial this implementation with both new and existing applications (via translating proxies when necessary). We also plan to use Haggle to develop and evaluate solutions to various challenges in PSN, including forwarding algorithms, security policies, usability aids, and resource management policies.

Acknowledgments

The authors would like to thank Augustin Chaintreau at Thomson and Richard Gass at Intel for insightful comments in discussions on this work. This work has been supported by the European Union under the Haggle integrated project FP6-IST-027918, and by the ACCA coordination action FP6-IST-6475.

References

[1] Gartner, Inc., "Market trends: Usb flash drives," http://www.gartner.com/DisplayDocument?doc_cd=130007.
[2] D. D. Clark, "The design philosophy of the DARPA internet protocols," in *Proceedings of ACM SIGCOMM (Computer Communications Review Vol 18, No 4)*, 1988.
[3] 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 the SIGCOMM 2005 Workshop on Delay-Tolerant Networking (W-DTN05)*. ACM.
[4] A. Lindgren, A. Doria, and O. Schelén, "Probabilistic routing in intermittently connected networks," in *Proceedings of The First International Workshop on Service Assurance with Partial and Intermittent Resources (SAPIR 2004)*, August 2004.
[5] J. Leguay, T. Friedman, and V. Conan, "Evaluating mobility pattern space routing for DTNs," in *Proc. INFOCOM*, 2006.
[6] R. Gass, J. Scott, and C. Diot, "Measurements of in-motion 802.11 networking," in *IEEE WMCSA 2006 (to appear)*, 2006.
[7] D. Clark and D. Tennenhouse, "Architectural considerations for a new generation of protocols," in *ACM SIGCOMM*, 2000.
[8] P. Hui, A. Chaintreau, R. Gass, J. Scott, J. Crowcroft, and C. Diot, "Pocket Switched Networking: Challenges, feasibility and implementation issues," in *Proceedings of the Workshop on Autonomic Communications*, ser. LNCS, vol. 3457. Springer-Verlag, 2005.
[9] S. J. Rahul C Shah, Sumit Roy and W. Brunette, "Data mules: Modeling a three-tier architecture for sparse sensor network," in *IEEE Workshop on Sensor Network Protocols and Applications (SNPA)*, May 2003.
[10] M. A. Wenrui Zhao and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," in *ACM Mobihoc*, May 2004.
[11] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of human mobility on the design of opportunistic forwarding algorithms," in *Proceedings of IEEE INFOCOM*, 2006.
[12] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of ACM SIGCOMM*, 2003.
[13] Delay Tolerant Networking Research Group, "http://www.dtnrg.org/."
[14] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," *Lecture Notes in Computer Science*, vol. 2009, 2001. [Online]. Available: citeseer.ist.psu.edu/clarke00freenet.html
[15] R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM 2001*, San Diego, CA, September 2001.