Data Management in Human Networks

Calicrates Policroniades
Telenor R&I
Fornebu, Norway
calicrates.policroniades@telenor.com

Pablo Vidales, Martin Roth, Daniel Kreienbühl
Deutsche Telekom Laboratories
Berlin, Germany
pablo.vidales@telekom.de
martin.roth02@telekom.de
daniel.kreienbuehl@epfl.ch

ABSTRACT

In this paper we study the use of a semantically rich storage model in order to fulfill the data transmission requirements of challenged networking environments, which are characterised by long delays and frequent communication disruptions. Practical experience shows us that the highly successful data abstractions of mainstream storage systems (e.g. monolithic file representation) operate poorly in emergent networking environments such as Delay Tolerant Networks (DTNs); short contact times do not allow for complete file or bundle transmissions. We have ported and integrated two systems in order to provide a solution that overcomes many of the data transmission challenges of DTNs: a semantically rich storage system (Datom) and a network framework capable of exploiting this augmented expressive power (Haggle). Our solution, Bedouin, enables both systems to run on resource-constrained devices. It facilitates *meaningful* data exchanges in challenged networks supporting the principle of infrastructure-independent networking, and exploiting human mobility and opportunistic connectivity. The design and function of a proof-of-concept Bedouin-based peer-to-peer file sharing application for human networks, called Caravan, is included. Experimental results demonstrate that our solution enables applications to work correctly in spite of intermittent data exchanges and disruptions while maximising the amount of useful data delivered to applications.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Network communications; E.2 [Data Storage Representations]: Linked representations.

General Terms: Design.

Keywords: challenged networks, data management.

1. INTRODUCTION

In this paper, we put forward the integration and strong cooperation of the network, storage functionality, and applications to overcome the challenges posed by data exchange in DTNs [7,22]. Unlike traditional infrastructure-based communication systems, data transmission in DTNs suffer from long and variable communication delays and arbitrary periods of link disconnection. We consider that the current networking and storage abstractions [4, 11, 15] suffer important limitations when taking into account the data transmission requirements of resource-constrained devices working on DTNs.

In the networking context, the way in which data encapsulation and fragmentation is performed by the network makes it unaware of the semantics of the data being transmitted and, thus, unable to cooperate with overlaying applications. At the application layer of the network stack, application data is mostly seen as a flat stream of bits to be fully and reliably transmitted to a destination node.

Moreover, additional constraints are posed by the storage abstractions. The file API handles applications' requests in a generic way, which is mainly concerned with storing application data as blocks of bytes and with managing a reduced set of associated attributes [11, 15]. From a networking perspective, writing code supported on the file abstraction suffers various shortcomings. Any *semantic knowledge* about applications' persistent abstractions that is present at run time is lost within the data stream representing the file data. Data manipulation through the file API greatly assumes access to a *file as a whole*, or in the best case to opaque regions of data within the file.

With these two shortcomings, it is impossible for the network to prioritise which part of the data stream should be transmitted; an important disadvantage in the context of DTNs since connections may die suddenly without completing the transmission of the whole data stream. Corrupted data has to be discarded, or in the best case stored to be reconstructed using a subsequent data exchange.

The lack of flexibility of the network to effectively fulfill applications' data transmission requirements in challenged environments creates the need for a mediator. With the integration and support of a more evolved data model, DTN networking architectures could improve data transmission tasks based on the synergy between the network and applications. If the appropriate data abstractions are used, cooperation can be enabled and meaningful data units can be efficiently transmitted across temporal paths based on fine-grained data access, and data transmission prioritisation according to the application and user preferences.

Although other storage abstractions, such as those offered by XML file formats [10,21] and databases [27], may be used to provide augmented levels of abstraction and fine-grained persistent data access, they introduce overhead and pro-

CHANTS'07, September 14, 2007, Montréal, Québec, Canada.

gramming models which make their usage impractical; since efficient use of resources such as energy, bandwidth, memory, storage, and CPU cycles, is important when resource-limited mobile devices are considered.

The rest of this paper presents Bedouin, our attempt to provide a solution to the difficulties stated above. Bedouin incorporates ideas from two existing systems: Datom [18] and Haggle [22]. Datom is a semantically rich storage layer that moves from the traditional view of file content as a monolithic element by exposing the structure, relationship, and type of persistent data in a systematic way. Haggle represents a network framework capable of exploiting the augmented expressive power provided by Datom. Thus, the contribution of our work is the integration of existing concepts in the areas of persistent data management and networking in a novel manner.

Sections 2 and 3 of this paper introduce Datom and Haggle, respectively. Section 4 presents their integration into Bedouin; both systems have been ported and adapted to work on resource-constrained mobile devices. We have tested the Bedouin implementation using Caravan, a peer to peer file sharing application for human networks, which was installed and deployed on a collection of mobile phones; this evaluation is presented in Section 5. We analyse related work in Section 6. Finally, Section 7 concludes this paper and presents potential research directions for this work.

2. AN ABSTRACT VIEW OF DATA

Because of the characteristics of data transmission in DTNs (e.g. short contact times, common disruptions and disconnection, intermittency, and delays) the interaction of the network and storage layers is of fundamental importance. The network's incapacity to understand any degree of abstraction in application data brings on many limitations. For example, the network is incapable of acting on its own to transmit application-meaningful portions of information in an opportunistic way or to prioritise which part of a data stream should be transmitted if connectivity is limited, or intermittent. Also, corrupted data streams have to be commonly discarded, or in the best case stored to be reconstructed using a subsequent data exchange, which might only take place much later.

The file abstraction has many limitations in the context of DTNs. The file API handles applications' requests in a generic and untyped way, which is mainly concerned with storing application data as blocks of bytes and with managing a reduced set of metadata [11, 15]. Any degree of structure, type, or semantic knowledge about applications' persistent abstractions that is present at run time is lost within the data stream representing the file data.

The file abstraction greatly assumes access to a file as a whole, or in the best case to opaque regions of data within the file. Since a file represents a monolithic object, data transmission takes place in an *all-or-nothing* fashion. This is an important disadvantage in the context of DTNs since connections may die suddenly without completing the transmission of the whole data stream. In the best scenario, it is an application's concern to identify which sections of the file are present in the data stream, keep track of their location in the file, manage sharing of file content, and provide the semantic meaning of data.

This situation can be improved if the level of abstraction used at the storage layer is augmented and made explicit to the network. Or in other words, it is necessary to *expose* structure, type, and persistent data access semantics, to the network in a systematic way. With this purpose we have decided to take concepts from Datom [18], since we consider it offers unique advantages over other storage technologies.

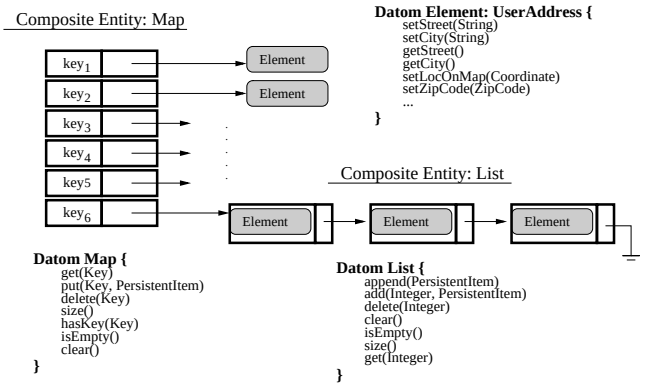

Figure 1: Example of the relationships and interfaces of persistent items in some of the abstractions of the Datom data model.

2.1 The Datom Storage Layer

The Datom storage system follows two principles to improve the interaction with persistent data: expose application-specific **data organisations** and bring to light persistent **data types** through components with well-defined interfaces. These principles are embodied in the two main types of persistent items of its data model: Composite Entities and Elements.

- **Composite Entities** are persistence-capable data structures which exhibit access semantics of a group of commonly used abstract data types. These data structures correspond to the nodes in the graph of persistence of the Datom data model. There are five types of Composite Entities, each of them supports different data access semantics. These are **stack, queue, list, map, and matrix**.

- **Elements** are the fundamental unit of storage for application data in the Datom storage model. Datom Elements can be regarded as docking points at which application programs load or store typed data. They represent application-specific programming abstractions with rich semantics and defined access routines. Accordingly, the definition and specification of Elements is based on applications' abstractions.

The Datom storage layer manages application data as a rooted graph of persistence made of these two types of items; it is correct to say that Composite Entities provide structure and access intent while Elements contain actual application data. Figure 1 shows a hypothetical piece of persistent data holding addresses; in which data is organised using a *Datom map* as an entry point. To get access to any of its members a system should use the well-defined map interface. Actual addresses are contained in the application-defined UserAddress Element. In addition, Composite Entities can be combined

to create more complex data representations. This is illustrated in the figure in the last member of the map which contains a *Datom list* of addresses; probably a user with more than one address. To access these addresses a program will have to use list semantics. Once a UserAddress is recovered, its data is accessed using application-defined routines (e.g. `getStreet()`).

The decision to employ a set of abstract data types (i.e. stack, queue, list, map, and matrix) as the primary interface to a storage layer follows one empirical observation: they are the most common data structures used by applications to manage persistent data objects at run time. There is practical evidence that they are commonly employed at runtime to organise persistent data layouts [18]. Because these abstract data types are commonplace in programs, implementations of them abound in modern programming languages as native types or as additions into their standard libraries [1, 16, 25]. In the context of databases, they have been used as fundamental storage elements [17], as a support tool to manage databases' query results [3], or as an interface to the underlying database data model [26].

Building the Datom storage layer on top of these abstractions represents a *minimalist* approach to the issue of getting access to the abstract composition of file data. They afford the possibility of representing application data more accurately and systematically expose persistent data semantics and applications' access intent. By exploiting this augmented level of abstraction, the network is able to understand persistent data layouts and manipulate a graph of persistence made of basic building blocks that exhibit well-known interfaces.

The main objective of Datom Elements is to use application-specific data semantics to manipulate persistent data. Elements are type managers that aim to provide a light-weight mechanism to store application data. As far as Datom is concerned, the data items stored inside a given Element lack identity and are managed through the interface of the holding Element. Elements' data is thought to be transmitted together (also clustered on disk) since they clearly reflect the data access semantics of the application, and as a result, a powerful hint to spatial locality of reference.

Some features in the implementation of Datom were especially important for us. The design and implementation of Datom weighs the trade-offs between functionality, overheads, and generality. Datom is enabled with incremental data loading based on the navigation that applications perform on the graph of persistence, which represents a mindful use of resources based on the application's access patterns. Furthermore, it provides automatic data movement between the volatile and persistent data spaces, automatic memory management, full control of update granularity, and atomic updates. Describing in detail these features is out of the scope of this document. In section 4 we will, however, elaborate on the aspects that are relevant to our purposes.

The Datom storage layer opens the possibility of performing application-wise data transmission strategies, and thus, improving the network interaction with persistent data. We consider the Datom storage layer to be a good solution for three main reasons:

- It enables the network to systematically disclose structure and exploit the advantages of fine-grained data access in an application-meaningful way.
- With Datom, application data is able to describe itself at a fine-grained level. Then, the network is able to identify application data, and also to associate metadata such as security, and delivery information to pieces of information.
- It substantially improves the cooperation and communication between applications, the network, and the storage system by sharing a common understanding based on high-level abstractions.

Certainly, other storage technologies could also have been used in place of Datom to provide augmented levels of abstraction and fine-grained access to persistent data. However, we considered important the following issues in our decision. Although a considerable amount of persistent data of applications running on top of file systems is amenable to structural decomposition, its access pattern [19,29] does not map properly to database functionality [27]. Furthermore, applications that use databases accept as necessary the overheads introduced by a high-level query language (e.g. SQL, OQL, XQuery) such as parsing, query optimisations, access path and plan selection, and query execution in exchange of the ability to dynamically query the content of the database. Efficient use of resources such as energy, bandwidth, memory, storage, and CPU cycles are important constraints in challenged environments. Datom has been, from its onset, conceived as a light-weight alternative; it disregards the type of data access facilities that may represent unnecessary overheads [18]. This, in our case, is especially convenient since we deal with resource-constrained devices on DTNs.

The manipulation of XML file formats has limitations in the context of DTNs as well. The programmatic APIs used to process XML data depend heavily on stream-based data manipulation to disclose file structure and data types; this reproduces many of the limitations observed in traditional file processing. In pull based APIs, such as SAX [21], data is processed sequentially, backward data navigation is not possible. This is a major shortcoming in challenged networks, an environment in which data exchanges are intermittent and retransmissions are commonplace. Tree-based APIs, such as DOM [10], create whole-file in-memory representations and thus are able to provide tree-like navigation. However, this model puts great strains on system resources, especially if the XML document is large. When compared with XML APIs, Datom aims to systematically expose data type and structure without relying on internal file formats by employing a rich set of programming abstractions equipped with fully navigational and incremental loading capabilities.

3. HAGGLE NETWORKING

Within the framework of the EU-project Haggle, a layerless architecture for mobile devices has been developed [22]. Haggle is a clean slate approach that solves many of the disadvantages of using the original fixed networking model in highly dynamic mobile environments. The key idea behind Haggle is to release applications from the concern of transporting applications' data according to the underlying network. Haggle architecture enables applications to work independently of the transmission method employed, e.g. ad hoc neighbouring communication, fixed network, or cellular network.

The four core concepts that allow this independence from the network layer are: data persistence support, flexible use

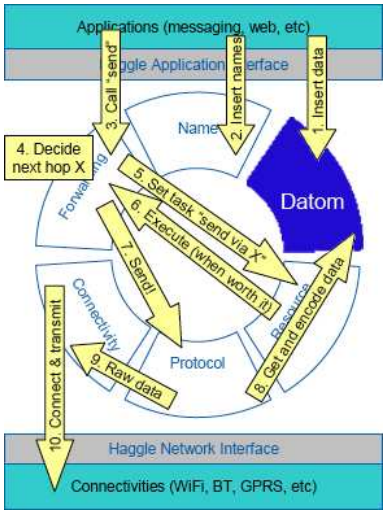

Figure 2: Haggle architecture and an example message exchange among Managers, when sending data. Adapted from [28].

of different networking protocols, name graphs supporting late binding, and centralised resource management. In Haggle, user's data is made visible and searchable to other nodes. Haggle includes support for multiple networking protocols which are employed dynamically. Name graphs, which relate user-level names to low-level addressable names, are independent from networking protocols to avoid making assumptions about the underlying infrastructure. Late binding is supported to avoid defining the data path a priori. Finally, resource management is integrated to have a single entity controlling the resources and not multiple applications making decisions based on individual views of the system resources.

The four aforementioned concepts are used to design a layerless networking architecture that comprises six modules known as *Managers*, which are illustrated in Figure 2. Haggle does not pass signals up and down between layers as the current OSI networking stack. Instead, managers have well-defined interfaces to communicate with each other, increasing the flexibility of the framework. In terms of the current networking model, Haggle removes the concept of layer completely, as opposed to breaking it as in the case of "cross-layer" solutions. In practice, applications request data transfers by creating a Data Object (DO) and a set of Name Objects (NOs), including the name graphs. The Forwarding Manager adds metadata about the forwarding operation using a Forwarding Object (FO), and all this information is communicated to the Resource Manager to be used in cost/benefit decisions.

An exhaustive description of the Haggle architecture is out of the scope of this document. However, the functions of each of the six managers in the Haggle architecture are introduced in the following paragraphs for readability purposes.

Resource Manager: It coordinates the activities of each Haggle manager. All outgoing or incoming network operations in Haggle are controlled by the Resource Manager. Decisions are taken on the basis of costs and benefits analysis. The Resource Manager is an example of the layerless nature of Haggle, as it receives tasks from many other managers such as neighbour discovery (Connectivity Manager) or e-mail checking (Protocol Manager).

Name Manager: To allow late binding without querying infrastructure services (e.g. DNS), name graphs are proposed [28]. Thus, endpoint description in Haggle is not done in the usual way of nested headers (e.g. MAC address, Ethernet address, IP address, TCP port, etc.).

Connectivity Manager: It maintains awareness of local connectivity on all interfaces. This manager encapsulates a number of Connectivity Objects (COs) that interact with the underlying hardware to provide communications functionality. COs include information about estimated cost (in terms of money, time, etc.) that is communicated to the Resource Manager.

Protocol Manager: It is responsible for encapsulating a set of protocols (e.g. HTTP, FTP, and SMTP) by which data can be forwarded. Haggle allows communication between the Protocol and the Forwarding Manager to send data (see Figure 2). In contrast, in the traditional networking stack application-layer protocols do not have direct impact on the forwarding decisions (i.e. network layer).

Forwarding Manager: It provides an API to applications to cause data to be sent remotely, it encapsulates forwarding algorithms (e.g. n-epidemic), and sends information about forwarding tasks to the Resource Manager.

Data Manager: It provides access to the local file system to the network and controls applications' data persistence. Haggle's data format is designed around the need to be structured and searchable. Relations between data units should be representable. The Data Manager presents an API to applications that allows them to easily search data according to user-level specifications.

The API that Haggle presents to the application is composed of a subset of the APIs that each Haggle manager provides to each other. The modular architecture of Haggle makes possible to contribute to the overall framework by replacing or extending the managers, an strategy that we have used to include the Datom storage layer into the Haggle framework, as depicted in Figure 2.

4. DATA MANAGEMENT IN HUMAN NETWORKS

The rationale of Bedouin, our proposal for *improving data management in human networks*, is the result of a process that can be summarised as follows. First, we studied a suitable architecture for human networks. We chose the Haggle framework due to its unique characteristics, clean slate design around mobile users, and modular architecture. Then, we identified the shortcomings of the current interaction between the network and the storage layer; supporting the opinion that if applications' data is to be visible and searchable by other nodes in the network then the use of high-level abstractions is compelling. Hence, we borrowed concepts from Datom.

The original Haggle Data Manager and its data model also include features such as data persistence management and metadata; it currently uses Java's standard interface to SQL and MySQL as its back-end storage for its Data Manager. This means that the current prototype is limited to run on laptops. Nevertheless, a SQLite version was mentioned in

the latest Haggle reports [28] for a future light-weight version of the architecture. However, our goal in employing a different data model is to improve the Haggle Data Management for situations in which discerning applications' data semantics and access patterns directly from their persistent data layout represents a key advantage; human networks built from resource-constrained mobile devices is one of such situations. Our work goes along with the Haggle principle of extending and providing novel managers to the original framework.

The unique features and benefits of combining Datom with Haggle can be summarised as follows:

- **Structured data and metadata:** Haggle defines data objects to organise data in trees, relate attributes to branches, and run queries on the data using DO filters. Datom facilitates all these features while providing a diverse set of semantically rich programming structures. This, in turn, enables applications to create persistent data layouts that match in a better way their own data access semantics.
- **Explicit persistent data layout:** The network is offered with a data model which is explicit about applications' data organisation. Used in a clever way, applications can organise their persistent data layouts such that features like priority, access pattern, data granularity, and locality of reference, are communicated to the network layer directly from the persistent data layouts. This means that the network no longer needs application-specific libraries to understand, and take decision according to, the data transmission requirements of applications; which motivates the following point.
- **Ease-of-cooperation:** The disclosure of the semantics of application data through the Datom Data Manager interfaces facilitates the cooperation between applications and the network since data exchanges can take place without the involvement of applications. As a consequence, asynchronous networking actions are enabled as data semantics are available across the whole system, and non-synchronous activity between endpoints becomes possible. This empowers Bedouin nodes to access cached information at intermediate nodes in an improved manner.
- **A light-weight storage layer:** The incremental data loading capabilities of the Datom storage layer enables the network to work exactly with necessary amount of information at any given time. When an application opens a Datom Composite Entity only the skeleton of this object is loaded into memory since all its members will have surrogates in place of concrete data elements. Surrogates are replaced only when an explicit access occurs. This enables the network to load the full skeleton of a graph of persistence and then solve persistent data requests using a high-level view of the composition of persistent data.

The rest of this section describes the porting and extensions of Datom and Haggle. Then, we analyse the extension to the Haggle Data Manager with a *Datom-based Data Manager* that exposes the new data abstractions and interfaces to the rest of the Haggle components.

4.1 Bedouin on Resource-Constrained Devices

Nowadays, the most popular mobile computing device among humans is, with no doubt, the mobile phone. For this reason, we decided to implement Bedouin to run on a collection of mobile phones, a realistic setting for our purpose. In practice, to deploy a human network based on Haggle and Datom on mobile phones, we needed to port and adapt both systems to use J2ME CLDC; a specialised Java interpreter for mobile devices with limited resources.

The porting of Haggle began with the publicly available source code from [9] in July 2006. The functionality of this version is far and above what is necessary, or even possible, for the mobile devices in the testbed. For instance, the original source allows access to connectivity options such as GPRS, WiFi, and TCP/IP, has hooks for applications such as mail, news, and web proxies, and supports multiple message delivery algorithms. Furthermore, it is written using Java Standard Edition (J2SE). This version of Java is incompatible with the Java Micro Edition (J2ME CLDC), which is supported by the mobile telephones in the testbed. Although, the two languages are largely syntax compatible, some changes are indeed necessary.

In order to speed development and minimise the size of the program, all superfluous functionality was removed, leaving basic manager functionality intact, along with a file transfer application interface, and only Bluetooth connectivity. Many data structures were changed, as J2ME CLDC only supports basic types such as Vectors and Hashtables. Instances of sleeping threads were replaced with the logic of waiting and notifying threads. Although this technique implies a more complicated program structure, it allows the program to only act when events happen, rather than continuously checking for them, which consumes more resources.

Similar steps were taken in order to port Datom to the J2ME platform. Of course, many data structures were changed to the simpler ones that are supported, but the primary change was the replacement of interfaces to external databases for management of the data elements with the use of the J2ME-included Record Management Store (RMS). RMS is, for many mobile phone models, the only way to access persistent storage using J2ME CLDC due to security issues. Unfortunately, RMS is also slow (also device dependent). Additional data storage interfaces were created, such as by using volatile memory (e.g. RAM). Though this is not a permanent solution, it is much faster than using persistent storage for the duration of any tests.

4.2 Extending the Haggle Data Manager

This section describes the integration of Datom into the Haggle porting by replacing the Haggle Data Manager with a *Datom-based Data Manager* that exposes the new data abstractions and interfaces to the rest of the Haggle components. From a high-level view, to integrate the portings of Datom and Haggle, we wrapped the Datom Manager implementation and included it as a modular component that extends the Haggle porting. The Datom Manager and the Haggle porting were not further modified, as the integration was designed as an extension.

Figure 3 shows in detail the integration of Datom into Haggle. Two interfaces were added: one to allow communication between Datom and the Haggle managers and another to enable applications to use Datom data abstractions. The former extends the Haggle Data Manager enabling com-

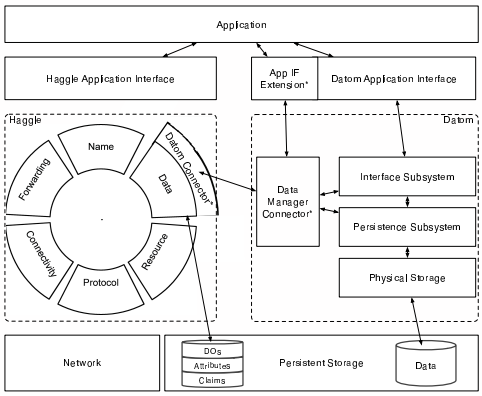

Figure 3: Datom improves Haggle by enhancing the data manager.

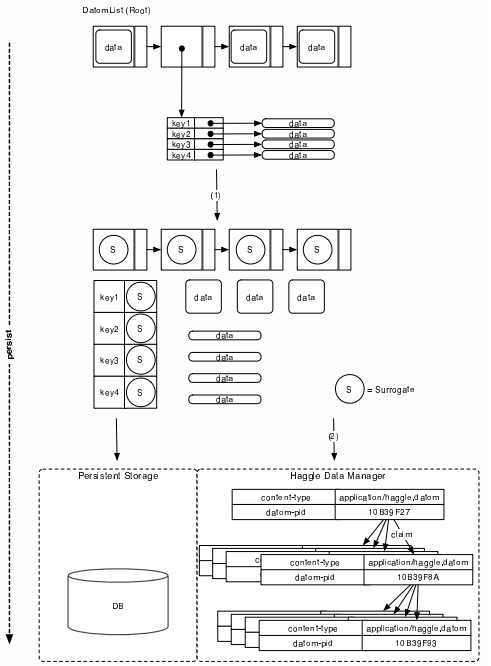

Figure 4: Datom and Haggle sequence to manage data persistence.

munication with Datom. Hence, the Data Manager can request Datom objects in order to send them across the human network. The latter allows applications to manipulate persistent data, in the form of complete graphs of persistence, a subgraph, or individual persistent items.

As shown in Figure 4, whenever a graph of persistence is saved, references to persistent items are replaced with surrogates, so that the graph can be split into individual components. Claims between these data objects are created accordingly, e.g. a DO referencing a Datom List would claim every DO that refers to the persistent objects in this list when its elements are traversed. The serialised form of persistent items is included as an additional attribute of the DO, when it is sent across the network. These properties enable incremental loading and facilitates networking with Datom graphs as it empowers Haggle nodes to reconstruct them as individual elements arrive.

Persistent items are uniquely identified by its Persistent ID (PID). As Datom was not designed for the purpose of networking, PIDs lack a user- or device-specific part (e.g. Haggle node name or a unique device ID derived from the Bluetooth address). In order to enable persistent items for networking, the identifier for persistent items was augmented to a globally unique identifier by including a device-specific part. Accordingly, as soon as an item is moved to persistence, Datom triggers the Haggle Data Manager to create a Data Object with an attribute (datom-pid) that contains the element's PID.

The Datom Data Manager closely interacts with the rest of the managers allowing efficient resource management in constrained mobile devices. Networking tasks take place according to user preferences, and data transfers observe aspects such as power, storage, and networking capacity usage. Fine-grained access to user data and data persistence control are key elements for this kind of well-informed resource management.

5. CARAVAN: A P2P APPLICATION FOR HUMAN NETWORKS

To illustrate the benefits of integrating Datom and Haggle, we implemented **Caravan**, a proof-of-concept peer to peer (P2P) file sharing application that distributes meaningful pieces of data across human networks. Caravan was implemented using J2ME CLDC and it runs on top of Bedouin. Figure 5 shows Caravan user interfaces.

Caravan performs incremental loading and exchanges meaningful data elements. In the prototype, people exchange photos (PNG format) using the Datom-enabled P2P application. The PNG files are decomposed into Composite Entities and saved using graphs of persistence. When two people are in contact, they exchange as many meaningful Datom objects as possible. Composite Entities are by default managed as of higher priority so that the destination can quickly reconstruct the skeleton of the Datom graph. The actual file data is transmitted using Datom Elements. We have decided to exchange photos mainly due to the visual effect of gradual image reconstruction, as shown in Figure 5. However, Caravan can be employed to transmit any kind of application data. In general, the definition of a particular persistent data layout is in charge of applications themselves, since it is their responsibility to hint, through the most suitable organisation of Datom items, their data semantics, access patterns, or transmission priorities.

Haggle nodes build the graph as Datom objects are received and mobile users can look at incremental versions of the photo as soon as they receive them. Composite Entities include multiple listeners for surrogates comprised in them, enabling applications to be notified upon reception of new persistent items. To enable Datom items for networking, the PID was extended to include a globally unique user/device identifier (e.g. unique device ID derived from a Bluetooth address).

Data exchanges happen based on *Caravan Interest Profiles* (CIPs), which contain Haggle node identifier, Datom root names, keywords, and PIDs of received and missing

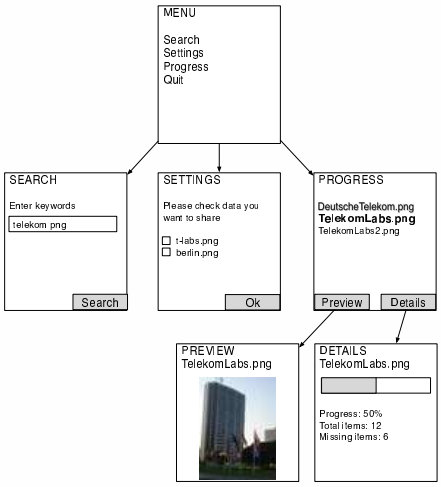

Figure 5: Caravan main user interfaces.

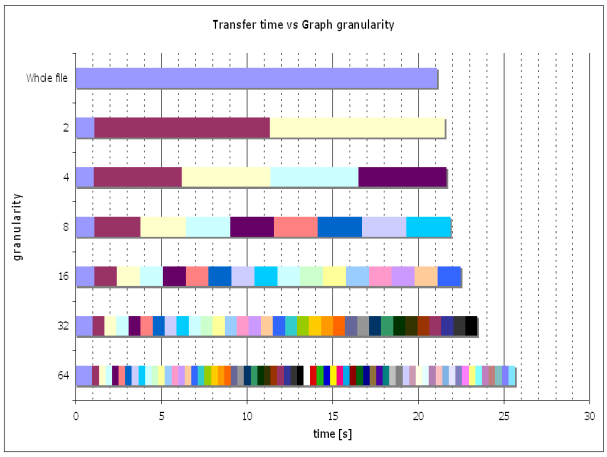

Figure 6: File Transfer Time vs. Graph Granularity

Datom objects. When two Haggle nodes in the human network have contact, they exchange profiles. This generates Caravan messages between sender and receiver. The sender is the node responsible to disseminate its Caravan Interest Profile. The receiver evaluates the message and forwards it to the *Profile Handler* that is part of Caravan. The Profile Handler compares keywords contained in the exchanged CIP with the Datom root names. Then, it checks for missing elements in the requesting node (i.e. sender).

There are two cases that originate from the exchange of CIPs. First, when a CIP-keyword matches a Datom root name, a message that contains the metadata of the Datom graph (Composite Entity) is sent to the requesting peer, together with one or more of Persistent Items. Second, in case the receiver has missing Datom objects, it sends them one by one to its peer node. Peers always store the most recent CPIs (using timestamps) to avoid handling the same data request more than once.

5.1 Experimental Results

The Caravan application exists only as a proof-of-concept demonstration. Measurement of the performance of the system as a whole is planned for the future. Caravan currently runs on a testbed consisting of Nokia 6630 mobile phones with 1 MB internal memory and J2ME CLDC, Symbian operating system, and Bluetooth stack implementation. However, the performance of sending Datom-ised files over a single link has been evaluated.

As shown in Figure 6, a 1 MB file is transmitted between two nodes, taking more than 20 seconds to be delivered in full. Then, we tested the positive effect of Data Element granularity. Each level of granularity is tested 30 times in order to ensure data consistency. As expected, meaningful portions of data are delivered much sooner, and ready to be used by the receiving node, than with the whole file approach. As the level of granularity is increased, a transmission overhead is incurred, as in this test each data element is sent by opening and closing a Bluetooth connection. The transmission time overhead is between 5% and 25%. This could be reduced by grouping many Data Elements together into single transmissions. Ultimately, after a short setup time, meaningful data can be delivered far before a monolithic file could otherwise be delivered. Short contact times can be easily taken advantage of, and data elements can be managed more flexibly.

Evidence that short contact times are a challenge in reality can be obtained from the analysis of contact times between WiFi users. Data available from the Crawdad Project [12], one of the most comprehensive studies of its kind, shows that in the Dartmouth College campus over the course of three years[1] roughly 55% of all intra-contact times lasted less than 20 seconds. Other networking interfaces such Bluetooth would presumably exhibit shorter contact times due to coverage and the nature of opportunistic human communications.

6. RELATED WORK

Most work in the area of DTNs and challenged networks focuses on proposing new architectures, as well as routing and delivery protocols [6]. However, less attention has been given to data persistence management for DTNs. Some of the concepts of data delivery in DTNs are taken from sensor networks, as these experience similar networking challenges. For example, DataMules [24] proposes an architecture to collect data in a scattered sensor network. The main concept is the use of mobile agents called MULEs that collect and transport information around the sensor network (e.g. vehicles, animals or humans). In a similar way, Message Ferries [30] are special mobile nodes that provide transport services to the rest of the deployed nodes in the sensor network. A delivery method is defined by the Bundle Protocol [23] that describes the format of the messages (called bundles) passed between DTN bundle agents that participate in a store-and-forward overlay network. Recent work has been published related to opportunistic content distribution in human networks [13]. Mobility traces were collected from a target group of mobile users to develop a content distribution model.

Previous research has examined the role of networked file systems in a mobile context [5,20]. These works assume, to a great extent, that there is a supporting infrastructure and a

[1]Dataset dartmouth/campus/movement/01 04 (v.2005-03-08).

stable link between a node and a particular server. However, we consider that the classical view of file data does not help to overcome the challenges of long delays and disruptions, or to fully exploit the opportunistic networking model.

Attempts to move away from the traditional file API to higher levels of abstraction have been investigated in different works, to a certain extent Datom shares the ideology behind them. Compound files in the OLE Structured Storage Model [2] represent a solution to the problem of internal file structure. Gribble et al. [8] explore distributed data structures and higher-level abstractions (e.g. hash tables, b-trees) as a storage infrastructure replacing the traditional flat view of data as a persistent data management layer. Under the same line, the Boxwood project [14] argues that higher-level abstractions can enable the system to perform better load-balancing, data prefetching, or informed caching.

7. CONCLUSION AND FUTURE WORK

Bedouin, i.e. the integration of Haggle and Datom, has been proposed in order to improve data management in DTNs. To practically asses the benefits of our approach and test our assumptions, we implemented Caravan as an example application. Caravan runs on Bedouin nodes travelling in a human network with short contact times and resource-constrained devices. Bedouin improves the networking model by making use of rich data semantics; during short contact times between nodes, Caravan is able to perform incremental data loading strategies and exchange meaningful portions of data. For future research, we plan to run tests in a realistic setting by installing Bedouin and running Caravan on a large number of mobile phones. We plan to deploy these devices in an office and report our results extensively. We consider that this paper can generate an interesting discussion on the open issues of data management in challenged networks during the workshop, a topic that has not been studied in detail in the past.

8. ACKNOWLEDGEMENTS

We would like to thank people at the Haggle project, and specifically to Jon Crowcroft and James Scott for the interesting discussions that enriched this work.

9. REFERENCES

[1] K. Arnold, J. Gosling, and D. Holmes. *The Java Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition, 2000.

[2] K. Brockschmidt. *Inside OLE 2*. Microsoft Press, 1994.

[3] R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[4] J. Crowcroft and I. Phillips. *TCP/IP and Linux Protocol Implementation: Systems Code for the Linux Internet*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[5] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch. The Bayou Architecture: Support for Data Sharing among Mobile Users. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 1994)*, 1994.

[6] Delay Tolerant Networking Research Group, 2002. Available at `http://www.dtnrg.org/`.

[7] K. Fall. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'03)*, pages 27–34. ACM Press, 2003.

[8] S. Gribble, E. Brewer, M. Hellerstein, and D. Culler. Scalable, Distributed Data Structures for Internet Service Construction. In *Proc. of the Symposium on Operating Systems Design and Implementation (OSDI '00)*, pages 319–332. USENIX Association, October 2000.

[9] Haggle Source Code, 2006. Available at `http://sourceforge.net/projects/haggle/`.

[10] A. L. Hors, P. L. Hégaret, L. Wood, G. Nicol, J. Robie, M. Champion, and S. Byrne. Document Object Model (DOM) Level 3 Core Specification Version 1.0. Technical report, World Wide Web Consortium, April 2004.

[11] S. R. Kleiman. Vnodes: An Architecture for Multiple File System Types in SUN UNIX. In *Proc. of USENIX Summer Technical Conference (USENIX '86)*. USENIX Association, 1986.

[12] D. Kotz and T. Henderson. Crawdad: A community resource for archiving wireless data at dartmouth. *IEEE Pervasive Computing*, 04(4):12–14, 2005.

[13] J. Leguay, A. Lindgren, J. Scott, T. Friedman, and J. Crowcroft. Opportunistic content distribution in an urban setting. In *Proc. of the 2006 SIGCOMM Workshop on Challenged Networks (CHANTS '06)*, pages 205–212, New York, NY, USA, 2006. ACM Press.

[14] J. MacCormick, N. Murphy, M. Najork, C. A. Thekkath, and L. Zhou. Boxwood: Abstractions as the Foundation for Storage Infrastructure. In *Proc. of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, pages 105–120. USENIX Association, December 2004.

[15] R. Nagar. *Windows NT File System Internals*. O'Reilly and Associates, September 1997.

[16] .NET Framework Developer Center, 2005. Available at `http://msdn.microsoft.com/netframework/`.

[17] Objectivity, Inc. *Objectivity/C++ Standard Template Library, Release 6.0*, August 2000.

[18] C. Policroniades. Decomposing file data into discernible items. Technical Report UCAM-CL-TR-672, University of Cambridge, United Kingdom, August 2006.

[19] D. Roselli, J. R. Lorch, and T. E. Anderson. A Comparison of File System Workloads. In *Proc. of 2000 USENIX Annual Technical Conference (USENIX '00)*. USENIX Association, June 2000.

[20] M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *In IEEE Transactions on Computers*, 39(4):447–459, 1990.

[21] Simple API for XML (SAX), 2005. Web site at `http://www.saxproject.org/`.

[22] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A Networking Architecture Designed around Mobile Users. In *Proc. of the 2006 IFIP Conference on Wireless on Demand Network Systems and Services (IFIP WONS 2006)*, January 2006.

[23] K. Scott and S. Burleigh. *Bundle Protocol Specification (IETF Internet Draft)*, May 2006.

[24] R. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *Proc. of the 2003 IEEE International Workshop on Sensor Network Protocols and Applications (SNPA 2003)*, May 2003.

[25] Silicon Graphics, Inc. *Standard Template Library Programmer's Guide*, May 2005.

[26] Sleepycat Software, Inc. *Berkeley DB Collections Tutorial*, September 2004.

[27] M. Stonebraker, L. A. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstain, and D. Beech. Third-Generation Data Base System Manifesto. *ACM SIGMOD Record*, 19(3):31–44, September 1990.

[28] 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 UCAM-CL-TR-680, University of Cambridge, United Kingdom, January 2007.

[29] W. Vogels. File System Usage in Windows NT 4.0. In *Proc. of the Symposium on Operating Systems Principles (SOSP '99)*, pages 93–109, New York, NY, USA, 1999. ACM Press.

[30] W. Zhao, M. Ammar, and E. Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proc. of the 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '04)*, pages 187–198, NY, USA, 2004. ACM Press.