Walkabout: Asynchronous Messaging Support for Mobile Devices

Adam Hudson and Bob Kummerfeld

School of Information Technologies
The University of Sydney, Australia
{ahudson, bob}@it.usyd.edu.au

ABSTRACT

We propose Walkabout, an asynchronous Internet messaging architecture designed to support data transmission between mobile devices with intermittent connections. A Walkabout-enabled device sends data messages via a proxy within its local network. The proxy creates a peer-to-peer overlay with other proxies in remote networks to deliver each message to the destination device(s). The overlay caches the message if a destination device is unavailable, migrates the message to follow devices as they move, and allows devices to resume uploads or downloads from different locations if interrupted. In this paper, we present the design of the architecture and its associated protocols. We present some potential applications, and show through simulation that Walkabout maximises local network use and provides significantly faster delivery times than the existing alternative methods under a range of device mobility patterns.

Keywords: mobility support, asynchronous messaging, network architectures.

1 INTRODUCTION

The processing, networking and storage capabilities of mobile devices are constantly improving. For example, Kodak's EasyShare-one range of digital cameras are equipped with wireless networking, Nokia's N93 provides video recording and wireless networking on a phone, and the Ultra-Mobile PC from Microsoft has all the power of a tablet PC in a tiny form factor. Devices such as these typically run software to backup, share or retrieve files across the Internet. While they do have the necessary hardware to enable these transfers, in reality there are a number of challenges that they face.

The first problem is that network connectivity is not always available, so direct transfer techniques are not always suitable. In particular, if the sender and receiver are both mobile, it may be difficult for both to obtain a connection simultaneously so that the transfer can proceed. Asynchronous messaging solutions such as publish/subscribe messaging, message queues and blackboards can help to overcome this, by providing an intermediate storage point that removes the requirement for simultaneous connection.

The second problem is that as a result of their increased capabilities, these devices may want to transfer large data files such as images, audio and video that can reach many hundreds of megabytes in size. Depending on the available Internet speeds, these transfers could take many minutes or even hours, so even if a mobile device has network connectivity, it may not always be practical or possible to maintain network connectivity long enough to complete them.

We propose an asynchronous messaging architecture, called Walkabout, that takes advantage of local network caches to support data transfers where one or more of the participants is a mobile device, and thus overcome these problems.

When a Walkabout-enabled *client* device joins a network, it locates a *proxy* server on the local network and then connects to it. The proxy accept messages from clients and transfers them to other proxies across the Internet, caching uploads and downloads in a way that allows clients to maximise their use of the network connection time available to them. If a transfer is interrupted, it can be resumed when the device reconnects. A peer-to-peer *message overlay* network is created for the delivery of each new message, incorporating all the proxies contacted by the sending and receiving devices. Messages are discrete and can range from small inter-application messages up to large media files of many hundreds of megabytes.

The main contribution of this paper is an architecture that makes Internet data transfers involving mobile devices practical, especially when connectivity is limited. We show that when the producer is mobile or the consumer is mobile and repeatedly visiting the same location, Walkabout reduces the amount of network connection time that a mobile device requires to complete a transfer, while simultaneously improving the end-to-end transfer speeds, in comparison to alternative techniques. The levels of improvement are proportional to how much faster the local wireless network is compared to the Internet link. The resultant system enables a number of applications that require large data transfers to and from mobile devices.

The remainder of this paper is organised as follows: Section 2 presents some application scenarios for Walkabout; Section 3 details the key system components while section 4 outlines the protocols and algorithms that drive it; Section 5 presents our evaluation of the system and its performance; Section 6 outlines the differences between Walkabout and similar systems; Finally, section 7 describes some future possibilities and concludes the paper.

2 APPLICATIONS OF WALKABOUT

This section presents some motivating scenarios for Walkabout, involving mobile uploads, mobile downloads and transfers between mobile devices. They illustrate some of the applications that become possible when there is a rapid form of asynchronous messaging available to a mobile device.

2.1 Upload From Mobile Device to a Server

Applications for the transfer of user-created content to fixed servers across the Internet are a major motivator for Walkabout, and figure 1 presents an example of this. In this photo backup application, a person takes their WiFi enabled camera with them on holidays. As they walk around taking photos, they come into contact with many wireless access points that provide a Walkabout proxy. The camera uses these opportunities to offload any new photos to the local proxies as quickly as the wireless network allows. If a transfer is interrupted, it is simply resumed at the next available opportunity. The proxies deliver the data across the Internet to a media server in the person's home, where the photos are stored permanently. The photos are now available in a stable location for family and friends to browse, and they can be deleted from the camera if more storage space is required.

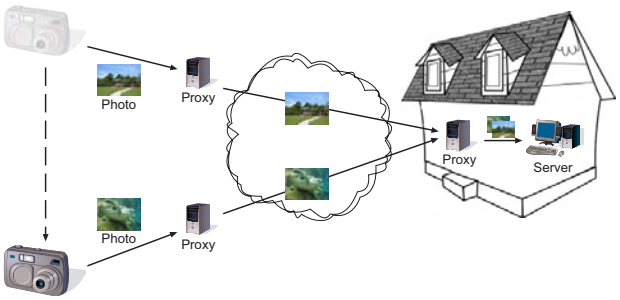

Figure 1: A camera uploads photos to a home server using Walkabout as it moves between networks.

These same principles could be applied to video, audio, or scientific data collection while in the field. Destination servers could be managed personally, by an organisation, or provided by a company for a fee.

Practical upload services have several benefits. Mobile devices are prone to loss, corruption and tend to have limited storage, so transferring content to a safe location protects the data, while simultaneously freeing up space on the device for more content. It also simplifies content publication, as devices can make their data available for others to access with only minimal network connection time required.

2.2 Download to Mobile Device From a Server

Portable media devices can often hold in the order of gigabytes of data, but this still tends to only be a subset of what a user owns or has access to. Walkabout messaging opens up possibilities for users with wireless multimedia devices to access remote files while on-the-move. A user can select a video file, for example, from a list of the contents of their home media server, then lodge a request across the Internet to download it. The server sends the file to their handheld video player, which downloads as much as possible each time it has Internet access, until the file is complete and ready for viewing. This same technique could also be applied to enable mobile devices to purchase media directly from online stores.

These applications could be extended to remove the need for requests, by incorporating modelling and smart environments. Take the example of a media server that records television shows. It knows that the user likes to watch the 7pm news, so it records the program automatically each night. However, if the smart environment detects that they are not at home, it actively sends the program, via Walkabout, to the mobile device that it decides to be the best, which could be their laptop, their phone, or any other device they regularly have with them.

2.3 Transfer Between Mobile Devices

The main advantage that asynchronous messaging offers for transfers between mobile devices is the ability to exchange data without being simultaneously connected. SMS and MMS are existing examples of this, but the ability to transfer larger files opens up the potential for audio and video messaging applications. The University of Sydney's Keep-in-Touch family messaging system [1] could be extended to incorporate mobile devices, for example.

Another application could be the trading of user-created game content, such as character models or maps, on portable gaming devices like the Nintendo DS or Sony PSP.

3 ARCHITECTURE OVERVIEW

Whenever it acquires a network connection, a Walkabout client device uses Apple's Bonjour[1] service discovery protocol to find a proxy and register with it. When a *producer* client wishes to send a message, it uploads a header describing the message contents and destinations to its proxy, followed by *pieces* of the message as fast as the local network will allow. The proxy uses the location-independent address of each *consumer* device to contact the proxy they were last connected to, which then downloads message pieces, stores them, and forwards them to the consumer if it is still connected.

All of the proxies involved in the transfer of a given message locate each other via a *message tracker*, and join together to form a peer-to-peer overlay. This can enable parallel downloads when multiple proxies have message pieces available, either as a result of client mobility or a message sent to multiple consumers. Figure 2 illustrates an overlay of three proxies, for the delivery of a message to three consumers.

Any time a client moves and connects to a new proxy, its registration causes messages delivered to the old proxy in its absence to migrate to the new location for download. Similarly, if the client moves while it is uploading, it is able to resume the transfer upon reconnection.

The roles of the system components are explained in more detail below, and the full details of the registration and transfer protocols are explained in section 4.

3.1 DOLR Service

Walkabout requires a scalable mechanism for proxies to contact clients and message trackers, and this is something

[1] http://www.apple.com/macosx/features/bonjour

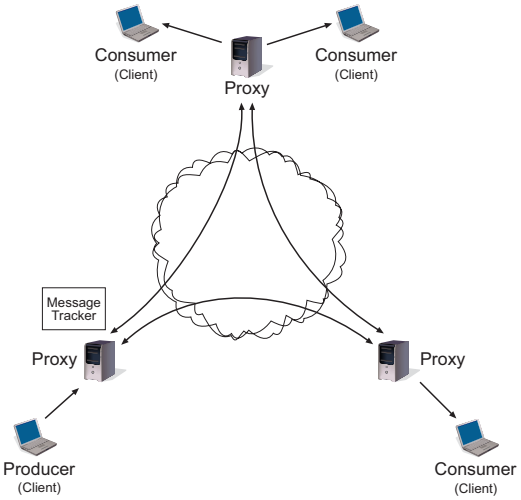

Figure 2: A Walkabout overlay network.

that a distributed object location and routing (DOLR) service such as Tapestry [14] can provide. DOLR enables a network node to associate a data object it holds with some globally unique identifier, such that other nodes can deliver messages to this identifier, and hence the node, in a scalable manner. Therefore every proxy joins a global DOLR service upon startup, which could be based on Tapestry or any other system that provides the same functions. Proxies publish entries when they accept a new client registration or create a message tracker in response to a new message, and other proxies route messages to them via DOLR. Hereafter, any time we say a message is *routed*, this means it was sent via the DOLR service.

3.2 Messages

The basic unit of message transfer is the piece. Messages are broken into fixed-sized pieces by the producer, the size of which are chosen on a per-message basis.

Every message transfer is preceded by a header, which the producer creates when it is ready to send the message. This header details the message properties, such as the total data length, piece size, piece hash signatures, and the identities of the consumers it is to be delivered to. The piece hash signatures allow nodes to check the validity of individual pieces after downloading them. Once created, a header is immutable, so the SHA-1 hash of the combination of all its fields uniquely identifies the message. This hash is henceforth referred to as the *message signature*.

Together, the proxies that have or seek pieces of a given message form a peer-to-peer overlay network that exists only for the lifetime of the message. Message trackers are the central store of information about an overlay, monitoring which proxies are overlay *peers* and the delivery status of each of the message's consumers. The first proxy to receive the message header from the producer creates the tracker, publishes the message signature to the DOLR service, and manages any subsequent updates. Overlay peers route requests and updates to the tracker using the message signature as the address.

3.3 Client

A client is an application that runs on a device and manages the transfer of messages to and from the overlay via its local proxy. The device may be fixed or mobile. Each client has an RSA key pair which it uses as the basis for identification and cryptography. In particular, the SHA-1 hash of the client's public key (or *key hash*) serves as a globally unique identifier. A proxy publishes a *client location* entry to the DOLR service whenever a client connects to it, associating the key hash with the proxy's own globally accessible IP address. This allows other proxies to send data to the client's last point of attachment by simply providing the key hash.

3.4 Proxy

A proxy is a network service that provides client connection, message transfer, caching and overlay maintenance functions. One would typically exist on a single dedicated machine within a private network, similar to a web proxy. It could be provided as a free service to trusted users in a home or office environment, as additional value to accompany other services, e.g., in a café, or as a wide scale subscription service deployed alongside wireless access points.

Proxies receive message pieces for upload from clients, and download them from their peers. Any pieces they acquire are held in a cache as long as possible, so that they may be delivered to clients connected locally, or downloaded by any other proxies that may need them.

Bonjour enables network service discovery both locally and across wide-area networks like the Internet. Therefore proxies implement Bonjour services, so that clients can find them within local networks, or remotely if necessary.

4 PROTOCOLS

A complete transfer requires several steps. First, the producer uploads some or all of a message when it is connected to a proxy. The overlay transfers this message to the appropriate proxies, and the consumers download the message from them. When all of the consumers have acknowledged or rejected the message, the transfer is complete. The following section explains how each of these steps work.

4.1 Client Connection

A client needs to find a proxy before it can access the system, so it searches the local network by sending out a Bonjour request. It connects if it finds a local proxy, otherwise it searches for a remote one. While it is preferable to use a local proxy, it is unlikely that one will be available in every network, so a remote proxy may be the only option at times.

When it accepts a new client registration, the proxy publishes the client's key hash to the DOLR service and contacts the client's previous proxy (at the address provided within the client registration). If it has any undelivered messages for the

client, the previous proxy sends across the message headers, and the transfer protocol is initiated as required.

4.2 Client Upload

When sending a new message, a producer first selects which consumers it wishes to deliver it to. Depending upon the application, the key hashes it selects could be pre-configured (e.g., for a backup server) or chosen from a list obtained via out-of-band means (e.g., from friends over email).

The producer initiates the upload to the proxy by transferring a header, then message pieces as fast as the local network will allow. Each time a piece upload is complete, the proxy verifies it against the hash in the header and acknowledges it if it is valid. This continues until the entire message is uploaded or the producer disconnects. If a transfer is interrupted, the producer sends the header again upon reconnection and resumes the upload from the first unacknowledged piece.

The proxy routes a request to the tracker to join the message overlay, or creates a tracker if it finds that none exists. It uses the information from the tracker to determine which consumers are still waiting for the message, then routes the header to each one's last location. The proxies that receive the header may then choose to initiate transfer of the message on behalf of their clients.

4.3 Transfers

Message transfers across a Walkabout overlay are inspired by peer-to-peer file transfer protocols such as BitTorrent [4]. They are driven by receiving proxies and make use of parallel uploads and downloads where possible.

Message states are communicated between nodes by way of *piece maps*, which are strings containing as many bits as there are pieces in the message. If a bit is set, it indicates that the proxy has that piece. These maps are generally quite small, e.g., a one gigabyte file transmitted as 256KB pieces generates a map of only 512 bytes.

The transfer negotiation begins when one proxy receives a message header from another. The proxy starts by finding which pieces, if any, its connected consumers want. It forwards the header to each of them, and expects a piece map detailing which pieces they want in reply. It assumes they want the entire message until it receives a response (which may never happen if the client has disconnected), and starts downloading immediately from the proxy that delivered the header. A consumer may also choose to reject the transfer completely, which leads the proxy to route a *rejection notification* to the tracker.

Next, the proxy joins the message overlay, and retrieves the list of peers and undelivered consumers from the tracker. It queries each peer to find what pieces they can offer, then downloads unique pieces concurrently from as many peers as possible.

Data transfers between proxies are performed in blocks, which are $\frac{1}{16}$ the size of a piece. Proxies pipeline requests, by requesting several blocks simultaneously from each peer. Once a proxy collects all 16 blocks of a piece, it verifies the piece against the hash value in the header. If successful, the proxy updates its download status to its peers, and delivers the piece to any connected clients that want it. Should the verification fail, whether due to a transmission error or a malicious peer, then the entire piece needs to be downloaded again.

A proxy uses a set of heuristics to select which pieces, and therefore which blocks, to request from which peers. Based upon BitTorrent, these heuristics aim to maximise overall network availability by requesting the rarest pieces first, but are augmented with additional rules to satisfy client needs as quickly as possible. A proxy continues downloading blocks on behalf of a client until it has all the pieces that the client wants, or the client connects to a different proxy.

4.4 Client Download

The final phase of a Walkabout transfer is the delivery of message pieces from a proxy to a connected consumer. If the proxy already has pieces of a message for the consumer in its cache when the consumer connects, the proxy delivers them immediately at local speeds. Once the unique pieces in the cache are exhausted, or if the consumer is connected when the message header first arrives, pieces are streamed as the proxy finishes downloading them. The client verifies each piece against its hash and acknowledges its receipt if valid.

Once the consumer has received all the message pieces it wants, the proxy routes a *delivery notification* to the message tracker.

4.5 Shutdown

A transfer is complete when every consumer has either downloaded or rejected the message. At this point, the producer and the peers need to be informed of the status, so that the tracker can be shut down and the overlay dismantled.

Peers route delivery and rejection notifications to the tracker as appropriate, which in turn forwards them to the producer. When the tracker knows that every consumer has either downloaded or rejected the message, it informs each of the peers in its list that the overlay is no longer needed. They acknowledge this, remove themself from the overlay and delete any pieces of the message they have in their cache. Similarly, once the producer knows that every consumer is accounted for, it routes a *producer shutdown* message to the tracker.

Once the tracker has received shutdown messages from the producer and all of the peers, it is no longer needed, so the proxy housing the tracker deletes it.

5 EVALUATION

To evaluate our model, we constructed a series of experiments using the OMNeT++ / INET framework [12]. Each experiment measures the comparative performance of the proxy-supported asynchronous delivery model of Walkabout, a direct synchronous model and a centralised server-based asynchronous model under different device mobility conditions.

The test network used in all the experiments contains 200 router nodes connected to a simplified Internet node by

100KB/s symmetric links with 10ms latency. Transfers across the Internet experience a delay proportional to the "distance" between the routers.

There are 64 consumer clients and 1 producer client in our network, and each one connects to a randomly assigned starting router over a 2MB/s link (simulating a good quality 802.11g connection). Our wireless model is simplified, as we are trying to isolate the mechanisms of our protocol from the effects of interference and signal strength, so a device always has a constant signal quality.

The Walkabout model has a single proxy connected to the router in each network via a 10MB/s Ethernet link. Any DOLR traffic is simulated by subjecting the message to an additional random 0.5-1.5 second delay during its delivery.

The centralised model also attaches a single server to a dedicated router with an effectively infinite Internet bandwidth and 1ms link latency, which clients register with when they connect to the Internet. The producer uploads its message directly to the server while it is connected, and the server forwards it on to the consumer when the upload is complete. In both this and the direct model, transfers interrupted through disconnection resume their exact position within the byte stream upon reconnection.

The direct model requires both the producer and consumer to be connected simultaneously for the transfer to take place. We assume that the producer always knows when and where the consumer is connected.

The main intended use of Walkabout is for the transfer of large files, particularly multimedia. Therefore, our experiments are run using either 5MB files, representing music files or high-resolution images, or 200MB files, representing half an hour of video. When the producer initiates a transfer, it randomly selects a consumer and starts sending the message.

Due to the symmetrical topology used in our simulations, the results obtained for each run are very similar, and hence have very low standard deviations of less than 0.5%, which has been omitted from the tables and figures.

5.1 Fixed Producer and Consumer

In this simple experiment, the end-to-end delivery times and application data overheads are measured when both the producer and consumer are fixed devices. 100 simulation runs were performed for each model using both 5MB and 200MB files, with separate runs for 256KB and 4096KB pieces under Walkabout.

While Walkabout is aimed at a mobile scenario, it is expected that it should still exhibit an acceptable level of performance in the static case, and our results support this. Table 1 shows the end-to-end delivery times for each of the models. Being the simplest, direct is the fastest for both file sizes, with centralised taking twice as long. Walkabout is only slightly slower than direct on average for both piece sizes. This is primarily due to the DOLR delay during the initial transfer establishment, and also due to the slight increase in overall traffic that Walkabout introduces. However, once the transfer is established, the pipelining of block requests makes it as efficient as a streaming protocol.

File size	Direct	Centr.	Walk./256	Walk./4096
5MB	53.7s	107.1s	57.2s	60.2s
200MB	2142.2s	4283.0s	2157.3s	2148.1s

Table 1: Average delivery time between static devices.

File size	Walkabout/256	Walkabout/4096
5MB	23.32KB (0.46%)	1.95KB (0.04%)
200MB	1018.39KB (0.50%)	59.45KB (0.03%)

Table 2: Average data overheads between static devices.

Table 2 shows the average data overhead per transfer. This measures the amount of application layer traffic that a Walkabout transfer introduces in addition to the actual message payload, including message headers, block requests, block headers and inter-peer progress updates. For the 256KB piece transfers, this amount is quite small, accounting for around 0.5% additional traffic. Increasing the piece size decreases the overheads even further to around 0.04%, as there are less individual messages required to complete the transfer and piece maps within inter-peer updates are smaller. It must be noted, though, that because a piece is the atomic unit of transfer between a consumer and a proxy, interruption to the client's connection will require any incomplete piece transfers to be restarted from the beginning. While the larger piece size is effective for static devices, it could lead to a significant amount of wasted transfer time if the client is moving rapidly. Therefore the piece size, which can be set on a per-message basis, should take these factors into account. We have shown here that 256KB gives acceptably low overheads, so we will continue to use it in all subsequent experiments.

These results confirm that, as a baseline indicator of performance, Walkabout is only marginally slower than a direct transfer when both endpoints are fixed.

5.2 Upload to a Fixed Server

This experiment explores the scenario from section 2.1 of a mobile device uploading to a fixed server. The mobile device follows a basic mobility pattern, where it connects to a network for a time, disconnects, then reconnects to a different network. The connection time for each run is fixed at a value between 10 seconds and 30 minutes, and the disconnection time is always 1 minute.

Figure 3 reveals that for a 200MB transfer, the direct and centralised models are badly affected by the shorter connection times, but that Walkabout is unaffected. This is due to the difference between the local network and Internet connection speeds. While connected, a Walkabout producer is able to upload data to its proxy faster than the proxy at the consumer's end is able to retrieve it across Internet. This creates a buffer of message pieces at the local proxy, which continues to provide data for download after the producer disconnects. The buffer is large enough that by the time the producer reconnects, it still contains data, and thus the transfer is able

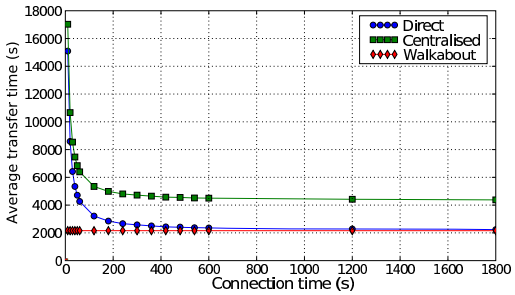

Figure 3: Average delivery times for 200MB between a mobile producer and a fixed consumer.

	Direct	Centralised	Walkabout
Upload time	2142.2s	2142.2s	105.1s

Table 3: Average total connection time required for a mobile producer to upload 200MB to the network.

to continue without interruption. The direct and Walkabout transfer times converge when the connection time is approximately 30 minutes, because the session time is long enough for the entire transfer to take place without interruption under both methods.

Even though the Walkabout and direct delivery times are quite similar after about 10 minutes, table 3 reveals that the Walkabout producer only needs around 5% of the time taken under the centralised or direct models to offload its data–which is the ratio between the local network and the Internet connection speeds. This means that a user uploading data from a Walkabout-enabled device is able to make much more efficient use of even the shortest of connection times, when compared to the alternative methods.

The data overheads are similar to those in experiment 1, and so are not presented here.

5.3 Download From a Fixed Server

This experiment represents the scenario from section 2.2, for a consumer device downloading a 200MB file from a fixed server. When a connection is available, the download to the consumer is simply a direct transfer for all methods, so there is little of interest to observe. However, different consumer disconnection times alter the amount of data that buffers at a Walkabout proxy in their absence, which can lead to improved transfer times if the consumer returns to the same proxy, but also to increased data overheads if they do not. Therefore the connection time in this experiment is fixed to 1 minute, while the disconnection time is varied between 10 seconds and 30 minutes for each set of runs.

The end-to-end delivery times for direct and centralised transfers would not be expected to change as a result of the consumer's movement patterns, but Walkabout is extremely sensitive to them. At one extreme, a consumer may always reconnect back to the same proxy, while at the other they might never visit the same proxy twice. We present these two mobility patterns in this experiment to explore the differences. In reality, the results for a transfer would be expected to lie somewhere between these two extremes.

5.3.1 Same Network

A consumer device could keep returning to the same proxy over the duration of a transfer if it is within a building or on a campus, but disconnect periodically due to fluctuating network coverage or intentional shutdown. In this scenario, the consumer-side proxy relays pieces to a connected consumer as it receives them, or buffers them if the consumer is absent. Upon reconnection, the consumer downloads the buffered data at local speeds, which offsets some of the time lost to disconnection. In fact, as figure 4 shows, when the disconnection time is low enough, no overall transfer time is lost. Even then, as the disconnection time rises, the main reason for increased delay is the time lost to disconnection itself. This claim is supported by figure 5, which shows that as disconnection time increases, the buffering at the proxy leads to a relatively small amount of time spent downloading by the consumer. The download time for the other methods are all the same, as they depend on the speed of the data supply from the Internet once the consumer reconnects.

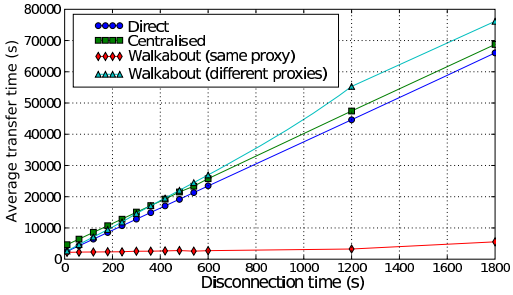

Figure 4: Average delivery times for 200MB between a fixed producer and a mobile consumer.

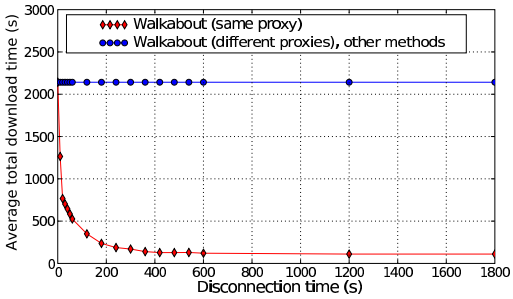

Figure 5: Average total connection time required for a mobile consumer to download 200MB from the network.

5.3.2 Different Networks

If the consumer connects to a different proxy each time, figure 4 shows that Walkabout's end-to-end delivery times are comparable to those of the direct transfer. However, figure 6

reveals that the data overheads increase proportionally with the increase in disconnection time, reaching over 2000% when devices disconnect for half an hour. By comparison, direct transfers are considered to have zero (or a very small constant) overhead and centralised delivery has a constant 100% overhead, due to the message being delivered in its entirety to both the server and the consumer. The Walkabout overheads are the result of the proxies continuing to download message pieces during the consumer's absence. If the consumer never returns to that proxy, the time spent downloading all these pieces is essentially wasted. It should be noted that the overheads only start to plateau when the disconnection periods are long enough that the proxy runs out of unique data to download.

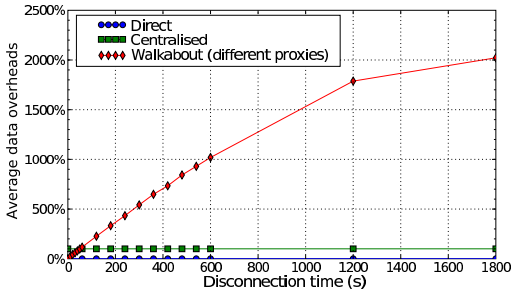

Figure 6: Average data overheads for 200MB between a fixed producer and a mobile consumer.

These overheads are very large, but they could be reduced by making some modifications to the Walkabout protocol. The simplest method is for the proxy to only download a message while a consumer is connected, but this eliminates the potential for speed benefits if the consumer does return to that proxy in the future. A better approach would be to selectively choose which consumers to download for. By recording how often it has seen a client, a proxy could decide whether the client is likely to return, and therefore whether it should continue to download pieces in their absence. The ideal approach, though, would be prefetching.

Prefetching has been shown to improve the performance of publish/subscribe messaging when there are multiple event broker servers, by migrating subscriptions for a disconnected host to the server where it is expected to reconnect [2]. By tracking the movement patterns of a client, the Walkabout network could predict which proxy a disconnected client will next connect to, and prompt it to begin downloading any messages that client is seeking. If the client does indeed connect there, it can begin downloading immediately and will experience a faster transfer than in the non-predictive system. With a good predictive model, this approach should yield near-optimal transfer speeds.

6 RELATED WORK

Asynchronous messaging solutions remove the tight coupling between communicating hosts, so that a producer may send data to consumers indirectly. Their store-and-forward nature allows a producer to place a message into the system without needing to know the addresses or even the identities of the consumers.

One form of asynchronous messaging has the producer storing its data at a rendezvous point for later retrieval by one or more consumers. A common example of this is email, where a message is addressed to an identity, delivered to a server, and stored until the recipient chooses to download it. A similar concept can be seen with tuple spaces, as demonstrated by Linda [6], where applications share data tuples by writing to and reading from a persistent, globally shared memory.

The publish/subscribe (or pub/sub) message-oriented middleware paradigm presented by systems like Elvin [9] and Siena [3] also removes the need for the producer to know the addresses of a message's recipients. Subscribers register their interest in certain events (and their current location) via subscription to an event broker, which may be a single server or a network thereof. A publisher generates a message and delivers it to the event broker, which forwards it as a notification to any subscriber with a matching subscription. The pub/sub paradigm does not intrinsically support disconnected operation, so if a subscriber is not connected to the service when a message is published, it does not receive it. However, there are several systems that introduce proxies to collect notifications even if the subscriber is absent.

Message persistence in the Java Event-based Distributed Infrastructure (JEDI) [5] is supported by proxy-like servers that are responsible for storing messages during a mobile client's absence, then migrating them to the client's new location if they connect to a different server. Walkabout proxies are similar, but do not require the client to explicitly say when it is disconnecting, as they do in JEDI. Elvin proxies [11] do not require explicit notification of disconnection, but they lack support for message migration between proxies.

The Java Message Service (JMS) [10] is a messaging API that supports both pub/sub and point-to-point message delivery models, with mechanisms that allow disconnected devices to receive messages upon reconnection. In the point-to-point model, consumers register with a central queue, a producer sends a message to the same queue, and it is forwarded to at most one of the waiting consumers. Pronto [13] and iBus//mobile [8] are JMS implementations specifically tailored to delivery to mobile devices across large networks (including the Internet). They provide network gateways which perform a variety of functions, most notably transcoding data to suit a device's limited capabilities, but do not place them in the same local networks as the individual communicating devices.

Data-caching infostations [7] support downloads in a similar manner to Walkabout. A mobile device requests a data object and pieces are delivered pre-emptively across the Internet to different infostations along its predicted path. The device downloads these pieces over a local wireless link when it is within range of an infostation. The infostations system does not include specific upload or inter-device messaging support, and so would not be able to support the ubiquitous applications presented in sections 2.1 and 2.3 without additional in-

frastructure.

Walkabout's proxy-to-proxy transfer protocol is based upon BitTorrent [4], where nodes query a tracker to find the location of peers for a given file, then request blocks from those peers. BitTorrent files are published without knowing who the consumers will be, and downloads are initiated by consumers actively seeking content that interests them. By contrast, a Walkabout producer specifies the consumers when it creates a message, so transfers are only initiated by those proxies trying to deliver the message to one of these consumers.

7 CONCLUSIONS AND FUTURE WORK

Current Internet data transfer solutions are not well suited to mobile devices. This is because they either require both sender and receiver to be connected simultaneously, or their transfer speed is restricted to that of the Internet. As a result, there are a number of data transfer applications that, while technically possible, are currently impractical. Walkabout introduces a proxy into the local network, so that mobile devices can communicate their data at local network speeds, but let fixed systems manage the transfer across the Internet. Proxies use peer-to-peer overlays to carry out these transfers, in a way that is tailored to the movement patterns of mobile devices.

We have shown through simulation that the Walkabout approach can significantly reduce the physical connection time required for a mobile device to upload or download a message. We have also shown that it can improve transfer speeds across the Internet when network connectivity is limited and either the message producer is mobile, or the message consumer is mobile and returning repeatedly to the same network. The amount of improvement is proportional to how much faster the local network speed is than the available Internet transfer speed. Our results prove that Walkabout would be an effective support infrastructure for any data transfer application involving a mobile device, including those presented in section 2. We have recently completed a prototype Walkabout system, and are currently in the process of developing these applications to test their real world performance.

A tracker permits the monitoring of message delivery status and overlay peers, but it does have time and data costs associated with overlay maintenance. These costs are acceptable for most transfers, but may be too expensive for smaller inter-application messages. We have devised a simpler delivery alternative that still provides applications with asynchronous delivery and location-independent addressing, and expect to include it in the main protocol in the future.

Finally, as suggested in section 5, prefetching could be a viable technique to apply to Walkabout. Of all the future directions, we believe that this holds the most promise for improving its performance.

ACKNOWLEDGMENTS

The authors would like to thank the Smart Internet Technology CRC for the support they provided to this research.

REFERENCES

[1] M. Assad, J. Kay, and B. Kummerfeld. The Keep-in-Touch system. In *Proceedings of Ubicomp 2005 Workshop on Situating Ubiquitous Computing in Everyday Life: Bridging the Social and Technical Divide*, Tokyo, Japan, September 2005.

[2] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, and M. Petrovic. Disconnected operation in publish/subscribe middleware. In *Proceedings of the 2004 IEEE International Conference on Mobile Data Management (MDM'04)*, Berkeley, California, USA, January 2004.

[3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design of a scalable event notification service: Interface and architecture. Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, August 1998.

[4] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, 2003.

[5] G. Cugola, E. Di Notto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27(9):827–850, September 2001.

[6] D. Gelernter. Generative communication in Linda. *ACM Computing Surveys*, 7(1):80–112, January 1985.

[7] D. J. Goodman, J. Borràs, N. B. Mandayam, and R. D. Yates. Infostations: A new system model for data and messaging services. In *Proceedings of 47th IEEE Vehicular Technology Conference*, May 1997.

[8] S. Maffeis. An introduction to wireless JMS. White paper, `http://www.softwired-inc.com`, 2001.

[9] B. Segall and D. Arnold. Elvin has left the building: A publish/subscribe notification service with quenching. In *Proceedings of AUUG97*, Brisbane, Australia, September 1997.

[10] Sun Microsystems. Java message service (JMS) API specification. `http://java.sun.com/products/jms/`.

[11] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGrid'01)*, Brisbane, Australia, May 2001.

[12] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.

[13] E. Yoneki and J. Bacon. Pronto: MobileGateway with publish-subscribe paradigm over wireless network. Technical Report UCAM-CL-TR-559, University of Cambridge Computer Laboratory, February 2003.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, EECS, UC Berkeley, April 2001.