Design Principles for Robust Opportunistic Communication

S. Keshav
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1, Canada
Email: keshav@uwaterloo.ca
TR CS-2009-35

***Abstract*—A mobile device can simultaneously increase its throughput and dramatically reduce energy and bandwidth usage costs by exploiting transient communication opportunities. We argue that this *opportunistic* communication mode will play a significant role in future mobile communication systems. We present some non-trivial applications that exploit opportunistic communication and their corresponding communication requirements. We outline the Opportunistic Communication Management Protocol, developed over the last four years at the University of Waterloo, that meets most of these requirements. We then focus on some design principles for robust opportunistic communication drawing from our experiences in developing and deploying several practical systems. We conclude with a sketch of some areas for future research.**

I. Introduction

With the proliferation of small, highly-functional wireless devices such as smartphones, embedded vehicular telematic systems, and wireless-enabled music players, it has become increasingly important to support *opportunistic* communication between a device and the infrastructure and amongst devices. In this mode of communication, data is exchanged over a wireless connection (between two devices, or between a device and a wireless access point) that lasts from a few tens of seconds to a few minutes. Surprisingly, even these short connection durations, if properly exploited, allow transfers of several tens of megabytes among the communicating parties [1]. This can enable innovative and useful applications, such as the distribution of audio and video content amongst people as they go about their daily routine–in effect, turning everyday social interactions into a low-cost and efficient content distribution network [2].

Although the potential for opportunistic communication has been known for a few years, building robust systems for opportunistic communication is surprisingly difficult. A system can be said to be robust to a fault if it can carry out its desired functionality, albeit with reduced performance, despite the fault. Robust system design, therefore, requires us to catalog a set of potential faults, and then prove, either by analysis or by actual test, that the system is robust to the fault. The designer of a robust communication system must anticipate and deal with many potential faults. The problem here is that, without actual deployment in the field, the set of possible faults is nearly impossible to determine.

Over the past four years, we have built a series of systems for opportunistic communication that we have deployed in the field and tested under realistic conditions. Based on our experience, we have identified several failure modes for opportunistic communication. These allow us to prescribe some principles for designing robust opportunistic communications, which are the primary focus of this paper.

The paper is laid out as follows. Section 2 presents some motivating examples of applications that exploit opportunistic communication. Section 3 presents the requirements for a practical system for opportunistic communication. Section 4 outlines the Opportunistic Connection Management Protocol (OCMP) architecture that meets these requirements. Section 5 describes the design principles in some detail. Finally, Section 6 presents our conclusions and directions for future work.

II. Some Applications

In this section, we present three examples of applications that can use opportunistic communication.

Consider a mobile device that has an on-board camera. Suppose that a user can take short videos using this camera and store it locally. Also assume that the device has a WiFi interface. The device could use this to opportunistically distribute content to similar devices. Specifically, each video clip could be annotated with a set of descriptive tags. On meeting another mobile, the devices could exchange tags of locally stored content. If one device wanted to obtain content matching certain tags, it could request it from the other. This would allow peer-to-peer wireless content dissemination. Clearly, the design of this application requires opportunistic communication. This example can be extended to allow flooding of 'want' lists to a set of nodes, and when a 'want' request can be satisfied, the routing of content back to the requester along an opportunistic path. It can also be extended to allow a set of nodes to maintain a completely distributed and self-indexed content database, similar to a peer-to-peer database, where content resides, is indexed by, and is mutually exchanged among a set of opportunistically communicating wireless nodes, as addressed by the Haggle project [2], [3].

A second example of opportunistic communication is to allow access to the Internet from moving vehicles [4]. Such vehicles could gain access to the Internet from roadside

wireless access points. Symmetrically, content created by the user of a mobile device could be placed into the infrastructure. For example, pictures or video clips taken by a mobile device user could be automatically uploaded to content sites such as YouTube or Flickr. Alternatively, mobile users could download e-mail and e-mail attachments as they drove past a wireless access point. By avoiding onerous charges for data transfer imposed by cellular providers, this mode of transfer makes it possible for device users to access rich multimedia content at low cost. This communication is opportunistic because vehicles lose connectivity as they move past the access point.

A third example of opportunistic communication is its use for rural communications. In this scenario, vehicles equipped with a wireless router and a small on-board computer exchange data with desktop computers in village kiosks [5], [6]. Packets transferred from the desktop to the vehicle are physically carried to other computers that provide a gateway into the Internet. The first such system was DakNet, which was built at MIT in in 2001 [5]. More recently, the VLink system from the University of Waterloo provides a similar service [7]. The use of opportunistic communication allows low-cost communication in areas where it is economically infeasible to deploy infrastructure. Even where such infrastructure exists, opportunistic communication augments traditional communication paths with a low-cost alternative. For instance, opportunistic communication can be used to maintain synchronization between a home and work desktop by means of USB memory stick (we describe this in more detail below).

In these examples the use of opportunistic communication enables applications that could not otherwise be implemented. Therefore, we believe that there is a need to support robust opportunistic communications for such future applications.

III. Requirements

In this section, we describe the requirements for any system that provides opportunistic communication. These requirements are derived from a careful consideration of the system support needed by the three applications presented in Section II.

We will assume, as a necessary prerequisite, that the applications are tolerant to both delay and delay variance. Otherwise, the applications would not be suitable for opportunistic communication in the first place. This is an important point: opportunistic communication is not feasible for all applications. It does not permit real-time communication or even interactive communication. It is best suited for moving large amounts of data where cost is a constraint and delay is not.

Here are the requirements for opportunistic communication:

- *Should not require human intervention:* to be useful, opportunistic communication should not require active participation of users who are likely to be engaged in other activities. Communication should proceed as a background process working on behalf of the user.
- *Should recover from disconnections:* Opportunistic communication necessarily implies that disconnections will be common. The system should recover from such disconnections, continuing existing data transfers from the point where they left off instead of starting them from scratch.
- *Should be low cost:* the system should make use of unlicensed spectrum when possible, to reduce costs.
- *Should be legacy compatible:* This allows easy deployment in legacy infrastructure. There should be minimal change to clients and servers and no change to underlying protocols such as TCP and IP.

In addition, we believe there are four secondary requirements which may not apply in all situations: to minimize device power usage, maximize use of communication opportunity, support both single and multi-hop communication, and provide over-the-air security.

These requirements cannot be met using standard TCP/IP. For example, on disconnection TCP goes into a series of progressively longer timeouts, and on reconnection, if the device acquires a new IP address, the communication state is completely lost. Moreover, TCP does not have any notion of communication cost. Therefore, it is as likely to use an expensive WWAN NIC as a free WLAN NIC. Given a choice of access points, TCP does not have any criterion by which to choose one. Of course, these requirements can be met by careful application design. However, this requires every application supporting opportunistic communication to implement the same functionality. Our goal instead is to design a standard set of primitives that can be used by broad class of applications.

It is difficult to meet these requirements primarily because of disconnections, which cause changes to every layer of the protocol stack. For instance, at the link layer, the presence of disconnections makes rapid WiFi association and agile selection of the data rate imperative. At the network layer, the routing protocol needs to take into account the fact that not all links are always available. At the transport layer, reliability cannot be achieved by timeouts and retransmission alone: every packet may need to be replicated for fault-tolerance. Finally, at the application layer, the API should encourage a send-and-wait programming style, rather than the current paradigm of request-reply that arises naturally from the socket API.

These changes and challenges motivate the design of a new protocol architecture for opportunistic communication that we call Opportunistic Communication Management Protocol (OCMP) and is described next [1]. OCMP is similar in spirit to the DTN Reference Implementation architecture [8] and Haggle [2].

IV. OCMP

OCMP is implemented at mobile devices and at a set of distinguished nodes, called 'proxies', that serve as always-available gateways for applications on clients to interact with legacy servers on the Internet (Figure 1). At its heart, OCMP

[1] An earlier version of the OCMP architecture is described in Reference [6]

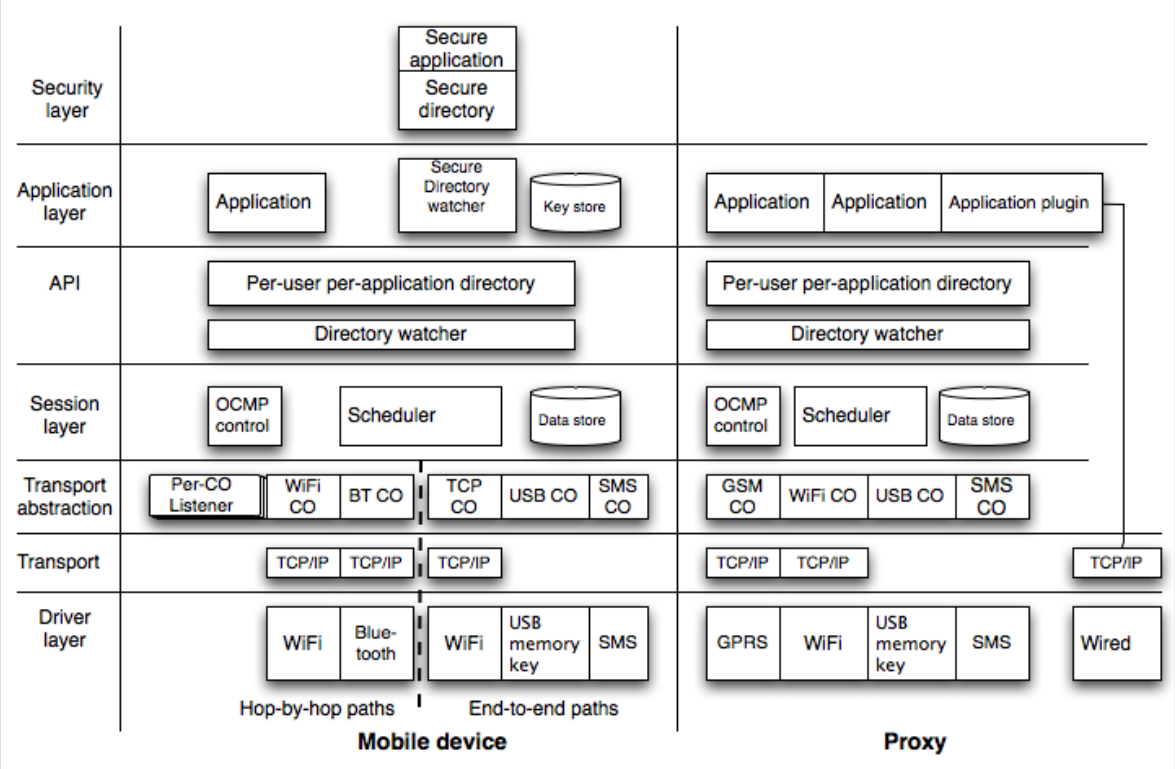

Fig. 1. OCMP software architecture

is a session layer that runs over both multi-hop paths and standard end-to-end (one hop) TCP connections to provide support for opportunistic communication. However, it is enhanced with an unusual application interface and support for end-to-end security, leading to some additional complexity, as described next.

We now describe the architecture layer by layer, starting at the bottom layer.

A. Driver layer

The lowest layer of the architecture is the driver layer. This corresponds to the device driver that provides a software interface to hardware network interface cards (NICs). In addition to standard communication interfaces, such as WiFi and wired NICs, OCMP supports some unusual driver types. The 'SMS NIC' allows data to be exchanged as SMS text messages and the 'USB memory key NIC' stores packets in the form of files on a USB memory key. These NICs are accessed directly by OCMP, without the use of TCP/IP. However, the data units sent on them (called *bundles*) are identical to those sent over other NICs. OCMP is able to use such unusual NICs because it does not assume the availability of end-to-end communication. Other NIC types, such as VSAT, WiMAX, or long-range WiFi, can easily be added to this architecture.

B. Transport layer

Transport over traditional NICs is provided by a standard TCP/IP transport layer. We do not make any modification to TCP/IP, thus allowing legacy compatibility.

C. Transport abstraction layer

The layer above the transport layer is the transport abstraction layer. We define a communication path to be session-layer generalization of a TCP connection that allows communication between two entities either over a single virtual hop, or over multiple hops, where each hop corresponds to a TCP-style connection. The communication path between two mobile devices, or between a mobile device and a proxy, is encapsulated by a connection object (CO) in this layer. Data given to a CO by a higher layer is transmitted and routed to the specified destination using the chosen path. We choose to map communication objects to paths because communication cost and delay is closely tied to the type of path.

Communication paths (and therefore, COs) fall into two broad categories: end-to-end paths and hop-by-hop paths. End-to-end paths allow a mobile device to communicate directly with the Internet. However, communication goes through an intermediate proxy for reasons such as disconnection tolerance and security. Examples of such paths are TCP over WiFi through an access point, corresponding to the 'TCP CO' (the TCP CO should really be called the 'end-to-end CO', but the name persists for historical reasons), an end-to-end SMS connection corresponding to the SMS CO, and packet transfer using an USB memory stick corresponding to the USB CO. In contrast, with a hop-by-hop path, data transfer goes from a mobile device to another mobile device. Examples are the WiFi CO and the Bluetooth CO, which use WiFI and Bluetooth respectively to communicate with a neighbouring node. There are similar COs for other end-to-end and hop-by-hop

TCP connections bound to each type of NIC (GPRS/EDGE, WiMAX, dial-up, etc.).

Note that the distinction between hop-by-hop paths and end-to-end paths is a matter of convenience and is used essentially as a routing hint. Bundles carry a destination name, which is either the name of a mobile device or of a proxy. These names are opaque to a CO. When two devices connect, they check if they have any bundles that can be given to the peer, either because the peer's name matches the bundle destination name, or because a routing rule indicates that this peer has a path to the destination. If so, bundle transfer happens. Because the same rules are used both by hop-by-hop and end-to-end COs, the distinction between them needed only to simplify the task of routing (as discussed in Section **??**.

COs are ephemeral and disappear when the corresponding communication opportunity is no longer available. Therefore, COs are instantiated and removed by a separate entity, called the *listener*. The listener uses CO-specific methods to learn of the presence of a communication opportunity and to instantiate a corresponding CO. For example, it can use WiFi beacons to instantiate a WiFi CO, or the presence of dial-tone to instantiate a dial-up CO. Listeners are also responsible for actually creating the connection between two COs. For end-to-end connections, the connection is always initiated by the client. For peer-to-peer connections, listeners set up persistent TCP servers, and periodically initiate connection attempts to a well-known port at each other's IP address.

We note here that some physical NICs, such as a WiFi NIC, support both hop-by-hop and end-to-end communication. That is, bundles can be sent over such a NIC either to another device or, in the presence of a WiFi access point, directly to the Internet. It is the job of the listener to distinguish between these two cases and instantiate the appropriate CO. Indeed, if the mobile device is in the presence of multiple access points, the listener sets up one CO corresponding to each usable access point, since each access point could support a different service quality.

D. Session layer

The OCMP session layer is responsible for scheduling application data units onto transports encapsulated using COs. It provides long-term database storage for application data from potentially ephemeral applications (i.e. from application that terminate after sending data). Finally, it exchanges control messages with peer session layers for neighbour discovery and bundle routing.

Unlike socket communication, where the loss of a communication socket is fatal, the OCMP session layer assumes that COs are ephemeral. Thus, it stores data, in the form of $bundles$, in a local file or database, and, when it is alerted to the availability of CO, establishes a connection path to the OCMP destination, and transmits bundles on the CO.

OCMP allows applications to specify, for each application-data unit, a prioritized list of COs on which it wishes to transfer data. OCMP will try to send data on the highest-priority CO that is available at that time. In an alternative implementation, applications specify a deadline by which time the data must be delivered and a scheduler within OCMP chooses the least-cost path that meets this deadline. In the latter design, the scheduler is augmented with a predictor, that is either told of, or can guess, future communication opportunities. The design of our system allows an application to use sophisticated scheduling policies should it so desire. For instance, an application can choose low-priority data to be sent using a TCP connection bound to a WiFi NIC, and high-priority data on the connection over a WWAN NIC. The design of robust scheduling policies is an open area of research. OCMP, however, provides the infrastructure for the implementation and experimentation with a variety of scheduling algorithms.

The OCMP routing protocol allows for both one-hop and multi-hop routing. As discussed earlier, COs are marked as end-to-end or hop-by-hop to facilitate routing. When a listener discovers the ability to use TCP to connect to a well-known port at the proxy, it instantiates a corresponding end-to-end CO. The routing module in the session layer then uses this CO to transfer bundles that are destined to the proxy (that is, have a destination name corresponding to the proxy). In the current implementation, all bundles to whom a hop-by-hop path is not known are also sent to the proxy for further routing [9].

If the listener cannot establish a TCP connection to the proxy, it creates a hop-by-hop CO and exchanges control messages with the peer session layer to determine the identity of the peer. It is now up to the routing module to decide which bundles should be transferred over this CO. In the current implementation, the routing protocol is naive flooding, so all bundles in the data store are sent to the peer. The design of more efficient hop-by-hop routing protocols is an open area of research.

E. Application programming interface

Applications communicate with OCMP using a 'directory API'. This essentially means reading files from and writing files to a per-user per-application directory. Files written by an application or plugin at a proxy to a communication directory are eventually transferred to a corresponding user directory on a client. In the other direction, files given to the proxy from a particular user are demultiplexed and written into an appropriate directory, and a handler designated by the plugin is invoked to carry out application-specific forwarding actions.

The directory API is implemented by a software component called a directory watcher. The watcher reads outgoing directories and, when it notices a new file, it fragments the file into bundles, saves the bundles in the data store, and calls OCMP to send the bundles. Similarly, it registers with OCMP to receive bundles on behalf of the plugins, and, when called by OCMP, re-assembles files, and when a complete file is received, writes the file into the application-specific directories. Applications can specify configuration parameters to the directory watcher by writing to a configuration file in

their communication directory. Details can be found in [6] or in online documentation [10].

F. Application layer

The application layer is at the highest layer of the OCMP stack. Recognizing the need for application-specific actions at a proxy, OCMP supports application plugins. Each such plugin is responsible for application-specific actions when OCMP receives an application-data unit. For example, plugins at the proxy are responsible for communicating with legacy Internet servers at Flickr, FTP, and YouTube. A plugin interacts with legacy servers using standard TCP/IP and hides client disconnections and IP address changes from them.

G. Security layer

OCMP provides an optional security layer. A 'secure directory watcher' application reads data to and from a 'secure directory.' When an application writes data to a secure directory, the watcher eventually notices it. It reads the destination field and uses this to access a local key store and retrieve the destination's public key. The data is then encrypted with a nonce and the nonce is encrypted with the destination's public key. This allows end-to-end secure communication.

Symmetrically, when encrypted data is received by an application, the secure directory watcher application is invoked and it uses the application's private key (stored in the user's file system hierarchy) to decrypt the data and place it in the secure directory. Thus, from the perspective of an application, to send data securely to a destination, all it needs to do is to write data to the secure directory, instead of to a standard directory. This makes it trivial to write secure applications.

For this to work, the key store needs to be properly set up. Our solution assumes the existence of a globally trusted third party that signs all public key certificates. These certificates are then flooded to key stores. This allows all mobile devices to know the public keys of all other mobile devices. Our current solution does not scale: the design of cheap, robust, secure, and scaleable security for mobile devices is an open problem. Details of our security architecture can be found in reference [11].

V. The Design Principles

Over the last four years, we have implemented the OCMP architecture several different times. Our first implementation only dealt with disconnections and supported only end-to-end paths [12]. In our second implementation, we added support for hop-by-hop paths, in particular, where this path was provided by a bus or a car that carried a router and provided 'mechanical backhaul' [6]. We also added support for security [11]. Our latest implementation, dubbed VLink, supports program state persistence across crashes and COs for USB- and SMS-based paths [7]. In parallel, the architecture has been independently implemented by Astilbe Corporation, a University of Waterloo spinoff, and is currently being used in the field. Therefore, we have accumulated a wealth of experience in the use of OCMP in the field. In this section, we present some design principles for robust opportunistic communication based on this experience. The principles are categorized according to the corresponding OCMP layer.

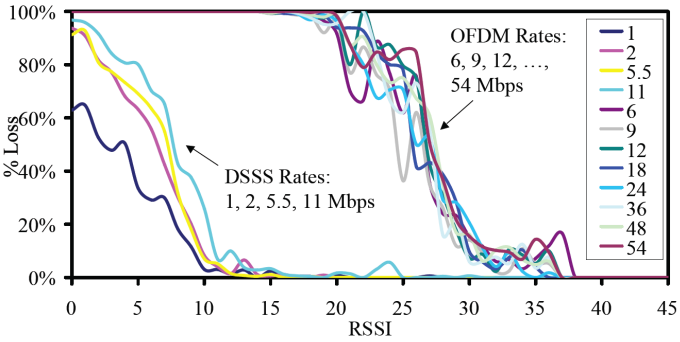

Fig. 2. Loss rate as a function of RSSI

A. Driver layer

Avoid the wireless fringe.

Opportunistic communication frequently uses wireless links such as WiFi and Bluetooth. These short-range technologies offer good throughput when the received signal to noise ratio is high. However, in their so-called "fringe", the signal-to-noise ratio is low and frame loss rates are high.

During opportunistic communication, two nodes that are initially out of range of each other come into range, exchange frames, and then move out of range again. Control frames–such as association and authentication request frames–sent when the nodes just enter each other's communication range are likely to be lost, triggering retransmissions and wasted communication capacity [1], [13]. The problem is exacerbated by the use of fixed multi-second timeouts in most device drivers. These timeouts nearly guarantee the loss of precious seconds of communication opportunity because of lost control frames.

These problems can be avoid by initiating communication only when the the received signal strength is high. Specifically, we found that loss rates during opportunistic communication decrease dramatically when the signal-to-loss ratio is greater than a threshold (which depends on the data rate), as shown in Figure 2.

The rule, simply, is to initiate transmission only when the signal strength exceeds this threshold. We found that, with this simple rule, throughput achieved during a communication opportunity can nearly double [1].

Note that this rule cannot be used in static scenarios, where the SNR may never increase past the threshold. In such cases, auto rate fallback is a better solution. But, with opportunistic communication, where mobility is a given, and the SNR is likely to improve as nodes move into range, deferring transmission in the hopes of encountering better SNR is the right decision.

Avoid performance coupling.

Performance coupling [14] refers to the dramatic decrease in performance across a good-quality wireless link from a mobile device to an access point because of the presence of a nearby mobile device that communicates with the same access point over a poor quality link. The problem arises due to 802.11's automatic selection of data rates. Nodes distant from an access point (AP) use a lower data rate to compensate for a lower signal-to-noise ratio on their communication path. This means that they take longer to transmit a single bit - up to 54 times slower than the best-possible bit transmission time in the case of 802.11g. Consequently, the overall performance of a BSS where some nodes have a lower communication capacity than others is dominated by the slower communication paths. This results in a single 'bad' node reducing performance for all other nodes.

To avoid this problem, we proposed that an AP service only those nodes that have good signal strength. This strategy, called MV-MAX [15], results in considerable performance improvement with only minor changes to the driver layer. The obvious problem with this solution is that it is unfair to those devices that are situated in a region with poor signal-to-noise ratio. However, because devices are mobile, it is likely that over time every device finds itself in a 'good' region. This is particularly true when dealing with vehicular opportunistic communication from a roadside access point, where every vehicle on the road goes through a poor quality region as well as a good-quality region, so that approximate fairness is attained.

B. Transport layer

Use hop-by-hop TCP.

In a network with opportunistic communication, end-to-end TCP is not always feasible [16]. Instead, we use hop-by-hop TCP over wireless links. The advantage of using TCP over UDP or raw IP is that provides flow control and reliability, which make it much more likely that transfers from one node to another succeed. Moreover, it exercises a heavily optimized and debugged path, which adds robustness. We did consider using our own implementation of an erasure code over UDP as an alternative to TCP, but the gains from this approach are yet to be proven.

C. Transport abstraction layer

Avoid wireless networks if possible.

Wireless networks, especially when used by mobile nodes, tend to be both unreliable and hard to debug. We found that the same device, moving through the same region of space in the vicinity of an access point, can experience dramatically different performance depending on the presence or absence of other devices and the unpredictable effects of multi-path interference. It is difficult to build a reliable system when the capacity of the wireless link can vary by nearly two orders of magnitude.

After considerable effort in trying to make wireless links reliable, we ultimately decided to provide, in addition to wireless links, a USB-memory key based communication path, where transfer between nodes occurs by physically carrying a USB-memory key from device to device. This eliminates problems with wireless networks. Because of the clean separation of functionality in our architecture, adding this solution required us to simply write a USB CO. The upper layers are indifferent as to whether communication uses wireless COs or the USB CO. Indeed, the USB memory key can be thought to hold 'frozen' packets, that are 'thawed' at the receiver.

This solution is far more cumbersome than one that relies entirely on wireless communication. However, in the context of rural communication, the added delay and loss of efficiency is minimal. Moreover, we no longer need a computer in a vehicle, greatly reducing costs.

D. Session layer

Use multi-copy routing.

A natural consequence of multi-hop opportunistic communication is that it is impossible to guarantee that a specific path exists between a source and a destination node. Moreover, node-to-node communication capacity is usually not a constrained resource. Therefore, instead of sending only a single copy of a bundle from a source to a destination, it is better to send multiple copies along more than one path - an approach called multi-copy routing [17]. Note that this does not apply for opportunistic communication between a mobile device and an infrastructure node, such as an access point.

The first version of our system used reverse path forwarding [6]. This is single-copy routing, and turned out to be efficient, but not sufficiently robust. The simplest approach to multi-copy routing is flooding. Here, a node gives a copy of every bundle in its data store to every other node that it meets. By exploring all possible paths in parallel, flooding-based routing is very robust. The current version of our system uses this approach.

The problems with naive flooding is that it is wasteful. To reduce its overhead, either the number of flooded copies, or the time-to-live of a packet can be adjusted [18]. Optimally choosing these parameters depending on the degree of connectivity in the underlying system is an open problem. Nevertheless, the use of multi-copy routing has been widely studied [19] and greatly increases the robustness of the system.

Use death certificates.

When the mobility pattern is unknown, naive flooding, although costly, is often the only robust solution, and indeed, the one we have adopted in our work. When node-to-node communication is cheap, the primary cost of flooding is in the storage of flooded bundles at intermediate nodes. There needs to be some way to eliminate these stored copies. One way is to remove bundles after some period of time, that is, on a timeout. The problem with this approach is that it requires setting the timeout value, which can be difficult to determine. If storage is not constrained, then choosing a conservative value for the timeout is a reasonable solution. However, there is a better solution.

Instead of timing out flooded packets, the destination of a bundle can acknowledge it by sending 'death certificates' [20].

These small bundles are also flooded and result in the removal of the corresponding bundle from the data store of a receiving node. Death certificates elegantly solve the problem of when to remove bundles. Moreover, when a sender receives a death certificate, it also gets a delivery acknowledgement, which is useful.

The use of death certificates leads to the recursive problem of how to remove the death certificates themselves. This is easily solved by keeping the size of a death certificate small and the time-to-live for a death certificate conservatively large.

Give higher priority to less-replicated data items.

When implementing flooding-based routing, we found that the same bundle could be transmitted during several communication opportunities. For instance, the arrival of every bus to a kiosk would result in the same bundle being transferred from the kiosk to the bus and $vice\ versa$. This is correct behavior, but results in a subtle denial-of-service problem: bundles that have not been sent could be queued behind bundles that have already been sent before, and may not be transmitted before the end of the communication opportunity, leading to persistent starvation.

Our solution is to prioritize those bundles that have been replicated the least. This avoids starvation and makes it more likely that they will make it to the destination. In practice, we store a transmission count with each bundle in the node database. We then query the database (using SQL) for all bundles that have been transmitted zero times, one time, and so on, using the results of these queries to determine the transmission order.

Use databases to store volatile state.

Both mobile devices and computers in developing regions can lose power, due to the battery running down, or grid power outages. On power loss, a typical session layer loses all program state and cannot known which bundles have been sent, how often they have been sent, and which bundles need to be acknowledged. To avoid this problem, the OCMP session layer maintains all data and state in a database, treating in-memory tables as a cache. Specifically, every in-memory table corresponds one-to-one with a database table. Access method to the tables are then modified so that the in-memory table and the database table are updated simultaneously.

In case the session layer crashes or the mobile device reboots, it refreshes all in-memory from the database. This makes it robust to power failures. We found this approach to be necessary in dealing with mobile devices and with deployments in developing areas.

This degree of robustness comes at a cost: writes are very expensive. Moreover, the amount of data that can be sent during an opportunistic communication is bandwidth limited by the database access throughput. We believe, nevertheless, that given the deployment environment, this is the right tradeoff to be made. In cases where the node is known to have access to reliable power, we can simply modify the access methods to not make database writes: a minor change in the code.

E. API layer

Consider using directory-based APIs.

Applications communicate with OCMP using a directory-based API. Files that need to reach a particular destination are placed in an 'upload' directory along with a metadata file that contains information that would normally be in a packet header, such as the destination, whether the file should be encrypted, and its priority. Any application-specific control information is also placed in the metadata file.

By using a file system as the communication API, application developers do not attempt to use a request-reply paradigm, which does not work in opportunistic communication networks. Instead, they adopt a 'send-and-wait' approach, which is more appropriate. We found that these hints to application programmers made it easy for them to write robust programs in a disconnection-tolerant environment.

F. Overall

Choose simpler solutions.

The first version of our system used Hierarchical Identity-Based Cryptography, flat names, Distributed Hash Tables, and a very general routing protocol [6], [21]. Through a process of simplification, we replaced these with PKI, hierarchical names, DNS, and bus-and-kiosk-specific routing protocols. This made our solution far more robust, though not so 'exciting' from a research perspective. In general, there is a tension between what we call 'full buzzword compliance' and building a system that actually works. We have chosen the latter.

VI. Related work

OCMP is a disconnection tolerant protocol, and is a practical implementation of the concepts first presented by Fall [16]. The work here is most closely related to two other implementations of the DTN concept: the DTN Reference implementation [8] and Haggle [2].

Both these implementations, like OCMP, store bundles in a local data store and offer both single-hop and multi-hop routing between nodes. They also incorporate ways to encapsulate different connection paths and offer support for experimenting with different routing and scheduling strategies.

Our work differs from the DTN Reference implementation in three significant ways. First, we bind connection objects to paths. In contrast, the equivalent to a CO, called a 'convergence layer', is bound to a protocol type, such as TCP or UDP. This prevents fine-grained control over scheduling and routing policies. Second, the reference implementation uses a socket-like API, instead of the directory-based API. Finally, the security model for the reference implementation offers the equivalent of link-level encryption unlike the seamless end-to-end encryption provided by our architecture. Moreover, it does not support a disconnection-tolerant mechanism for key distribution and management. Nevertheless, many of our ideas are derived from our long experience with using the DTN reference implementation and we owe it many insights.

Our work also differs from Haggle in some critical aspects. Haggle uses a non-layered architecture, where different agents

collaborate with each other to accomplish the forwarding task. In contrast, our approach uses traditional layering. Second, Haggle only supports multi-hop paths between devices and has no support for proxies or the Internet infrastructure. Finally, Haggle manages all data on behalf of applications: applications never own data. Again, we take a more traditional approach, where OCMP is responsible only for data transfer, not for application-level data management.

VII. Discussion

We believe that robust opportunistic communication requires careful attention to every layer of the protocol stack. In this paper, we describe our experiences in developing OCMP and in making it more robust. We hope that these experiences will prove useful to other researchers in the field.

There are still many open problems in designing, implementing, and deploying systems based on opportunistic communication. At the link or driver layer, there is a need for agile data-rate selection algorithms that can exploit the predictable increase and decrease in signal-to-noise ratio that is characteristic of opportunistic communication. There is also a need for association and authentication algorithms that are insensitive to losses, or, at the very least, do not respond to losses with a hardcoded long timeout.

At the transport layer, there is a need for adaptive erasure codes that are optimized for opportunistic communication. There is also the need for flow control algorithms that can limit data rates from transmitters in the presence of network congestion. The interaction of erasure codes with routing and flow control algorithms is likely to be both a challenging and an fruitful area of research.

At the session layer, a critical problem is to develop optimal algorithms for multi-copy routing over temporal graphs, that is, graphs whose topologies are time-dependant. There is the intriguing possibility of using centralized controllers to coordinate the actions of mobile nodes using lightweight control messages.

Finally, at the application layer, there is the need for non-trivial applications that can exploit opportunistic communication. Peer-to-peer media dissemination, distributed social networks [3], and vehicular communication networks [4] are examples of applications that have the potential to make the use of opportunistic communication nearly ubiquitous.

Acknowledgments

Figure 2 is due to David Hadeller and is from his unpublished research. This work draws from the talents of a host of graduate and undergraduate students at the University of Waterloo. In particular, I would like to thank Aaditeshwar Seth, Darcy Kroeker, David Hadaller, Matei Zaharia, Shimin Guo, Hossein Falaki, Usman Ismail, Earl Oliver and Mohammad Derakhshani for their many contributions.

References

[1] D. Hadaller, S. Keshav, T. Brecht, and S. Agarwal, "Vehicular opportunistic communication under the microscope," *Proceeding of ACM MOBISYS*, 2007.

[2] J. Scott, P. Hui, J. Crowcroft, and C. Diot, "Haggle: A Networking Architecture Designed Around Mobile Users," in *Proceeding of WONS*, 2006.

[3] A. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot, "MobiClique: Middleware for Mobile Social Networking," in *Proceedings of the SIGCOMM 2009 Workshop on Online Social Networks (WOSN)*, 2009.

[4] J. Ott and D. Kutscher, "A Disconnection-Tolerant Transport for Drive-thru Internet Environments," in *Proceedings of IEEE INFOCOM*, 2005.

[5] A. Pentland, R. Fletcher, and A. Hasson, "Daknet: Rethinking connectivity in developing nations," *Computer*, vol. 37, no. 1, pp. 78–83, 2004.

[6] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Low-cost Communication for Rural Internet Kiosks Using Mechanical Backhaul," in *Proceedings of ACM MOBICOM*, 2006.

[7] VLink project, Tetherless Computing Laboratory, University of Waterloo., "http://blizzard.cs.uwaterloo.ca/vlink."

[8] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra, "Implementing delay tolerant networking," *Intel Research, Berkeley, Technical Report, IRB-TR-04-020*, 2004.

[9] S. Guo, M. Falaki, U. Ismail, E. Oliver, S. U. Rahman, A. Seth, M. Zaharia, and S. Keshav, " Design and Implementation of the KioskNet System (Extended Version)," *University of Waterloo, Technical Report, CS-2007-40*, 2007.

[10] KioskNet, "OCMP Directory API," in *http://blizzard.cs.uwaterloo.ca/tetherless/index.php/OCMP_Directory_API*, 2009.

[11] S. Ur Rahman, U. Hengartner, I. Ismail, and S. Keshav, "Practical Security for Rural Internet Kiosks," in *Proc. of ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR 2008)*, 2008.

[12] A. Seth, S. Bhattacharyya, and S. Keshav, "Application Support for Opportunistic Communication on Multiple Wireless Networks," in *http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/05/ocmp.pdf*, 2005.

[13] Z. Zhuang, T. Chang, R. Sivakumar, and A. Velayutham, "A 3: application-aware acceleration for wireless data networks," in *Proceedings of the 12th annual international conference on Mobile computing and networking*. ACM, 2006, p. 205.

[14] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance Anomaly of 802.11b," in *Proceedings of IEEE INFOCOM*, 2003.

[15] D. Hadaller, S. Keshav, and T. Brecht, "MV-MAX: Improving Wireless Infrastructure Access for Multi-Vehicular Communication," in *ACM SIGCOMM Workshop on Challenged Networks (CHANTS)*, 2006.

[16] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proceedings of ACM SIGCOMM*. ACM New York, NY, USA, 2003, pp. 27–34.

[17] S. Jain, M. Demmer, R. Patra, and K. Fall, "Using redundancy to cope with failures in a delay tolerant network," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, p. 120, 2005.

[18] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. New York, NY, USA: ACM Press, 2005, pp. 252–259.

[19] E. Jones and P. Ward, "Routing strategies for delay-tolerant networks," *Preprint available from http://www.ieice.org/explorer/ITC-CSCC2008/pdf/p1577_P2-46.pdf*, 2006.

[20] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, "Epidemic algorithms for replicated database maintenance," in *Proceedings of ACM PODC*. ACM New York, NY, USA, 1987, pp. 1–12.

[21] A. Seth and S. Keshav, "Practical Security for Disconnected Nodes," in *Proc. First Workshop on Secure Network Protocols*, 2005.