Design and Implementation of the KioskNet System (Extended Version)*

S. Guo, M.H. Falaki, U. Ismail, E.A. Oliver, S. Ur Rahman, A. Seth, M.A. Zaharia, and S. Keshav
David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
{sguo, mhfalaki, uismail, eaoliver, surrahman, a3seth, mazahari, keshav}@uwaterloo.ca

ABSTRACT

Rural Internet kiosks in developing regions can cost-effectively provide communication and information services to the poorest sections of society. Yet, a variety of technical and non-technical issues have caused most kiosk deployments to be economically unsustainable [1]. KioskNet addresses the key technical problems underlying kiosk failure by using robust 'mechanical backhaul' for connectivity [2], and by using low-cost and reliable kiosk-controllers to support services delivered from one or more recycled PCs. KioskNet also addresses related issues such as security, user management, and log collection. In this paper, we describe the KioskNet system, outlining its hardware, software, and security architecture. We describe a pilot deployment, and how we used lessons from this deployment to re-design our initial prototype.

1. INTRODUCTION

Rural Internet kiosks in developing countries provide a variety of services such as birth, marriage, and death certificates, land records, and consulting on medical and agricultural problems. A typical kiosk has a Windows-based PC and a dial-up or VSAT connection to the Internet, and is operated by a computer-literate kiosk owner who maintains the system and assists end users. To effectively serve its users and be profitable to its owner, a kiosk should be highly available and should have a reliable connection to the Internet. Moreover, it should be low-cost, so that it can be sustained with a minimum of user fees. Unfortunately, due to limited electrical power, pervasive dust, mechanical wear-and-tear, and computer viruses, kiosk computers often fail, requiring frequent (and expensive) repairs. Similarly, network connectivity is often lost due to failures in the telephone system, inability to power the VSAT station, or loss of alignment of long-range wireless links. Faced with high costs and unreliable service delivery, customers quickly lose interest. Due to these factors, in addition to several other non-technical issues, kiosk deployments are often found to be unsustainable in the long term [1].

KioskNet attempts to make a kiosk more robust without increasing its cost, thus addressing at least the technical aspects that lead to lack of kiosk sustainability. It builds on two key concepts. First, it uses a single-board-computer-based, low-cost, low-power kiosk controller at each kiosk. The controller can communicate wirelessly with another single-board computer mounted on a vehicle (as was pioneered by Daknet [3]). These vehicles carry data to and from a gateway, where data is exchanged with the Internet. This 'mechanical backhaul' [2] avoids the cost of trenches, towers, and satellite dishes, allowing Internet access even in remote areas. In areas where dial-up, long-range wireless or cellular phone service is available, the kiosk controller can be additionally configured to use these communication links in conjunction with mechanical backhaul. Second, KioskNet allows refurbished PCs to boot from the kiosk controller. Kiosk controllers are reasonably tamper-proof so they offer reliable virus-free boot images and binaries. We do not use the PC's hard disk, thus avoiding hard disk failures and disk-resident viruses. Moreover, refurbished PCs are cheap and spare parts are widely available.

KioskNet has the following key features:

- The system is low-cost (see Section 4 for details) and appears to be economically viable. We estimate that our system requires a capital expenditure of $100-$700/kiosk, depending on the configuration[1], and an operating expenditure of $70/kiosk/month. These rough estimates include the cost of field technicians and capital depreciation. This is four to ten times cheaper than other solutions.
- The solution is rapidly deployable: we successfully installed a prototype in Anandapuram village, Vishakapatnam district, AP, India in two days during May 2006.
- Kiosk controllers are low-power (6-8W), therefore they can be run off a solar panel.
- Recycled PCs can run either the (Linux) binaries that are packaged with the kiosk controller, which are guaranteed to be virus free, or can boot into an existing operating system (typically Windows) from their hard drive for stand-alone computing.
- We can provide private and authenticated communication amongst kiosk users, and between a kiosk user and a secure node in the Internet.
- Our software is shipped in the form of a LiveCD that can be booted on any Windows or Linux PC. The CD is used to copy OS images directly onto hard drives, which are then installed in single-board computers.

*Technical Report CS-2007-40

[1]All figures are in US dollars.

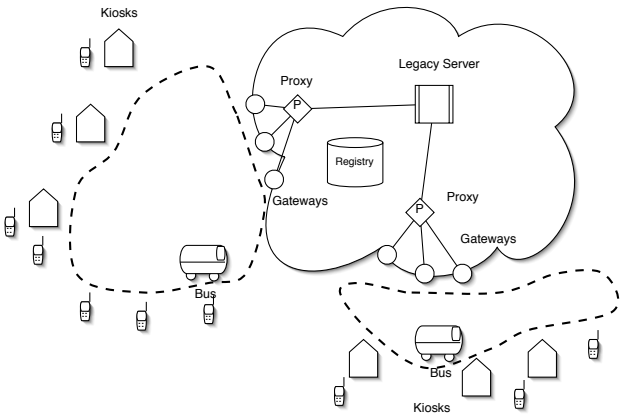

Figure 1: KioskNet overview.

- Our code is free under the Apache open-source license with no patent, copyright or intellectual property restrictions.

In the remainder of this paper, we present an overview of the system in Section 2 and its software architecture in Section 3. The security architecture is described in Section 3.4. We describe the cost structure in Section 4 and our experience with deploying the system in Section 5. Section 6 discusses some changes to our initial design decisions that reflect experiences from the pilot deployment. We present related work in Section 7 and conclude in Section 8.

2. OVERVIEW

KioskNet consists of a set of kiosks that use mechanical backhaul [2] as the primary means of communication to the Internet (Figure 1). *Ferries* carry data to and from a kiosk to a set of *gateways* that communicate with a *proxy* on the Internet. The remainder of this section describes these KioskNet components in more detail.

2.1 Kiosks

Each kiosk has a kiosk controller, which is a server that provides recycled PCs with network boot, a network file system, user management, and network connectivity by means of dial-up, GSM/GPRS, VSAT, or mechanical backhaul. A kiosk controller always has a WiFi NIC. In addition, for most deployments, we expect that kiosk controllers would also provide connectivity by other means, such as GPRS, SMS (GSM), VSAT, or a dial-up connection. Our current prototype uses headless and keyboard-less low-power single-board computers, such as those from Soekris Corp. and Via Corp., as kiosk controllers, although the controller functionality can be implemented in any commodity PC.

We would like kiosks to be used by two types of users. We expect most users to access the system from a recycled PC (also called a 'terminal') that boots over the network (using RAM disk) from the kiosk controller and can then access and execute application binaries provided by the kiosk controller over NFS. Recycled PCs cost approximately $100 and spare parts are widely available worldwide. Moreover, as a shared resource, they are an order of magnitude cheaper than any dedicated resource.

A second class of users, such as wealthier villagers, government officials, or non-goverment organization (NGO) partners, could access one or more kiosks, or a bus directly, using their own devices, such as smart phones, PDAs, and laptops. Such users could use the kiosk controller or bus essentially as a wireless hotspot that provides store-and-forward access to the Internet. Due to some technical issues (specifically, the difficulty in setting up public and private keys in such devices), the current distribution of our software does not support them. We intend to handle this in future releases of our software.

The set of kiosks in the same geographical area, and administered by the same entity, comprises a KioskNet *region*. Regions not only have administrative significance, in that all entities in a region are certified by the same certificate authority, but also have routing significance, because bundles are flooded within a region. Figure 1 shows a system with two regions, which could both be potentially managed by a single administrative entity.

2.2 Ferries

Although kiosk controllers can communicate with the Internet using a variety of connectivity options, our focus is on the use of mechanical backhaul. This is provided by cars, buses, motorcycles, or trains, that pass by a kiosk and an Internet gateway. We call such entities *ferries*.

A ferry has a single-board-computer that is powered from the vehicle's own battery. This computer has 20-40GB of storage and a WiFi network interface. It communicates opportunistically with the kiosk controllers and Internet gateways on its path. During an opportunistic communication session, which may last from 20 seconds to 5 minutes, we expect 10-150MB of data to be transferred in each direction. This data is stored and forwarded in the form of self-identifying *bundles*. Ferries upload and download bundles opportunistically to and from an Internet gateway.

2.3 Gateways

A gateway is a computer that has a WiFi network interface, storage, and an always-on connection to the Internet. Gateways are likely to be present in cities with DSL or cable broadband Internet access. A gateway collects data opportunistically from a ferry and stages it in local storage before uploading it to the Internet through the proxy. A region may have more than one gateway.

2.4 Proxy

We expect that most communication between a kiosk user and the Internet would be to use existing services such as email, financial transactions, and access to back-end systems that provide government-to-citizen services. Legacy servers that provide such services typically can neither deal with long delays and disconnections, nor easily modified. Therefore, we need a disconnection-aware proxy that hides end-user disconnection from legacy servers. We currently assume that there is one proxy per region.

The proxy is resident in the Internet and has two halves. One half establishes disconnection-tolerant connection sessions with applications running on the kiosk controller or on mobile users' devices. The other half communicates with legacy servers on behalf of disconnected users. Data forwarding between the two halves can be highly application dependent. To support application-specific communication

with legacy servers, we support *application plugins* at the proxy that coordinate their actions with a corresponding application at the kiosk controller or mobile device. For example, such a plugin implements SMTP to communicate with legacy mail servers on behalf of users at kiosks. The current release of KioskNet includes several proxy plugins, which are outlined in Section 3.9.

When the communication sublayer at the proxy receives application data from a plugin, the data needs to be transferred to gateway that is in communication with the destination kiosk. This is done using algorithms such as those described in [4]. The gateways subsequently hand off data to passing ferries for transport and delivery to a kiosk. The kiosk passes the data to an application specific plugin at the kiosk for delivery to kiosk users.

In the opposite direction, when a kiosk user wants to send data to the Internet, it is carried to a gateway, which transfers it to a proxy. The proxy passes received data to the associated plugin, which interfaces with legacy Internet servers. For instance, in the case of email, the proxy plugin would forward Internet bound emails using SMTP.

Besides serving as an application-layer gateway, a proxy provides a central point of management. It runs a DNS-based location register that is used for location management, as described in Section 3.3. It also maintains a $Whitepages$ database that maps from a user's globally unique identifier (GUID) to its X509 public key certificate. This database, which is replicated at each kiosk, allows secure communication among KioskNet users.

2.5 Legacy servers

The last component of our architecture, the legacy servers, are typically accessed using TCP/IP and an application-layer pro tocol such as IMAP, SMTP, or HTTP by a proxy. We do not require any changes to legacy servers.

3. SOFTWARE ARCHITECTURE

3.1 Communication architecture

KioskNet communication software runs on the following *components*: proxies, gateways, ferries (buses), kiosk controllers, terminals (recycled PCs), and cell phones (or PDAs). The overall communication architecture is sketched in Figure 2, which shows the software protocol architecture, and Figure 3 which shows the data path in the case where email is being sent from the kiosk to the Internet.

The communication system allows kiosk users to exchange messages with the Internet, other users in the same region, and with users in other regions. It also allows users to move to other kiosks in the same region, or kiosk in other regions, while continuing to send and receive messages. Finally, it allows users of mobile devices to use a kiosk to send and receive bulk data messages more cheaply than over the cellular phone network. This section presents a detailed description of the protocol architecture. We present the routing protocol in Section 3.2 and mobility support, including the structure of globally unique identifiers, in Section 3.3.

3.1.1 Protocol stack

KioskNet is an overlay network both on the Internet and on the cellular phone network. The base communication layers are (a) TCP/IP that runs on wired or wireless network interfaces and (b) the Short Message Service (SMS) present on the cellular phone network. KioskNet uses TCP/IP and the Internet to carry data, and SMS to carry control messages and potentially short, urgent, data messages. TCP/IP is present in all system components, and SMS is present in all components except for the terminals and possibly the gateways.

All components except for handheld devices and terminals also run the Delay-Tolerant Networking (DTN) overlay provided by the DTN reference implementation [5]. DTN provides a disconnection-tolerant end-to-end transport layer. Unfortunately, the reference implementation lacks some important features:

- It does not support selective flooding within a disconnected region.
- It does not support users who move from one kiosk to another.
- It does not provide the ability for a kiosk controller, cell phone, or proxy to use application-specific policies to choose from one of many network interfaces.
- It does not provide application-specific plugins at the proxy to allow seamless interconnection with legacy servers on the Internet.
- It does not support a cellular-phone based control plane.

To address these issues, we did two things. First, we modified the DTNRG's DTN 2.0 reference implementation to add selective flooding mobility support as described in more detail in Sections 3.2 and 3.3. Second, we designed and implemented the opportunistic connection management protocol (OCMP) [2,6], which runs on top of DTN and other available network connections. OCMP is a disconnection-tolerant, policy-driven session layer that runs over both DTN and standard TCP/IP communication paths, and allows applications to choose different network interfaces for different application-data units. It allows application-specific plugins to execute at the proxy to interact with legacy servers. It also provides a cellular phone network (SMS)-based control channel. Each type of available communication path is modeled as a connection object (CO) within OCMP. For instance, the DTN path is encapsulated as a DTN CO. There are similar COs for a TCP connection bound to each type of NIC (GPRS, EDGE, WiMAX, dial-up, etc.).

OCMP allows a policy manager to arbitrarily assign bundles to transmission opportunities on COs. This scheduling problem is complex, because it has to manage many competing interests: reducing end-to-end delay, while not incurring excessive costs, and maximizing transmission reliability. We do not know of an adequate solution to the general problem. Therefore, in the current implementation, we merely send application-specified 'urgent' data on an always-on connection (if that is available) and other data on the mechanical backhaul CO. We also allow applications to designate certain data items to be sent on the SMS CO. The design of our system, however, allows the use of more sophisticated scheduling policies without changing the rest of the system.

3.1.2 The SMS connection object

An interesting aspect of KioskNet's communication stack is its support for an 'SMS NIC', which allows communication over GSM cellular networks. Note that SMS is provided

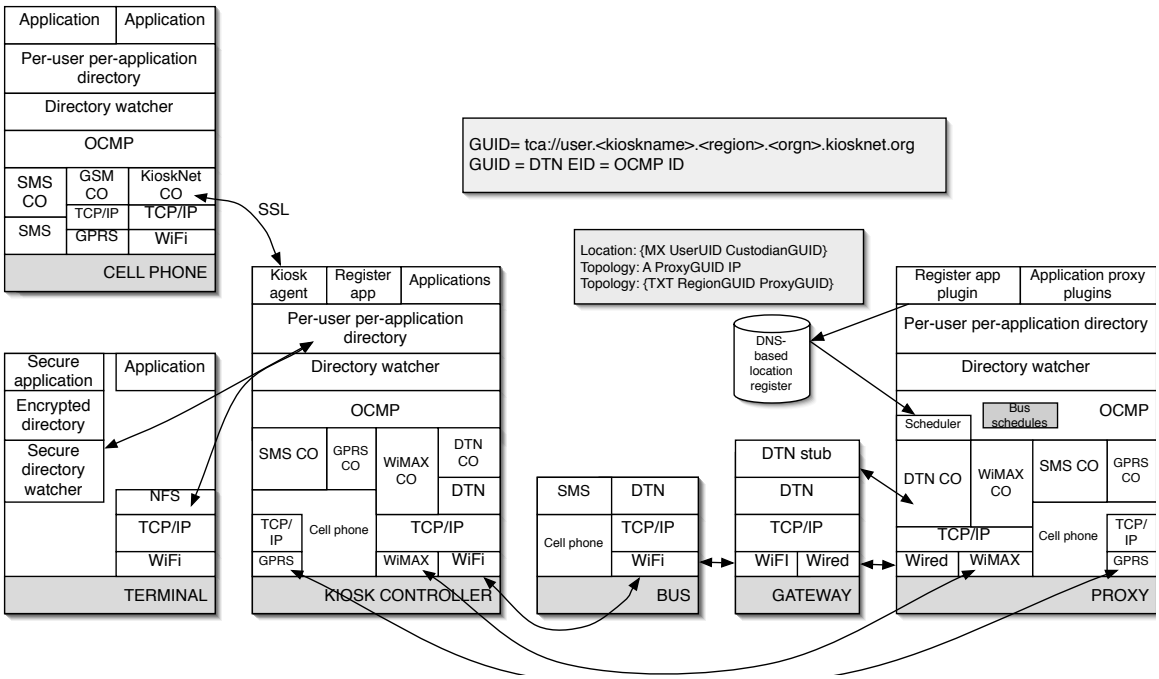

Figure 2: Software architecture.

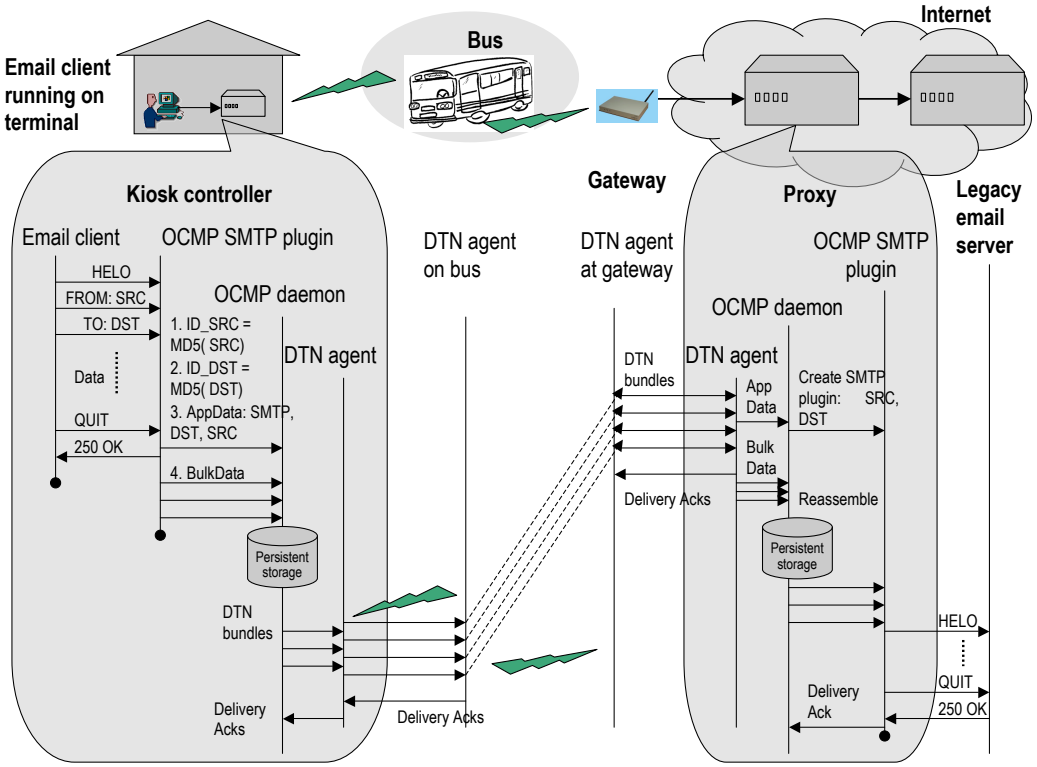

Figure 3: KioskNet data path when used to send email from a kiosk to the Internet.

natively by the GSM voice subsystem and does not require a user to subscribe to (typically expensive) data services. Messages passed to the SMS CO are fragmented into small (approximately 150 byte) pieces and sent as SMS messages to another SMS-enabled KioskNet component (which could be another kiosk controller, a ferry, or the proxy). The receiving component then collects the SMSs and reassembles the original message. Although SMS is known to be slow and limited in message size, it provides a low-bandwidth and low-delay control channel.

This has many potential uses, which we are only just beginning to exploit. For example, the current GPS position of a ferry can be sent as a single SMS message to the proxy. This would allow us to alert operators to ferry failures. We can also use the channel to send notifications of message arrival to be sent to kiosk users. This removes the need for users to frequently check for new email and other data, reduces overall contention for a kiosk, and allows a single kiosk to serve more users.

We now examine the protocol architecture in more detail. It is easiest do so 'right-to-left', that is, starting at the proxy, and working our way back to a terminal or cellular phone.

3.1.3 Proxy

Working our way down from the top (and, for the moment, ignoring the 'register' application plugin), note the presence of one or more application plugins. Each such plugin is responsible for communication with a legacy server on the Internet. Examples of such plugins are those responsible for communicating with Flickr, FTP, and YouTube. A plugin interacts with legacy servers using standard TCP/IP. To communicate with OCMP, however, it uses a 'directory API'. This essentially means reading and writing data to and from a per-user per-application directory. Data written by a plugin to a communication directory results in eventual creation of an OCMP bundle that is then sent, using the best possible means, to a user. In the other direction, data arriving to the proxy from a particular user is demultiplexed into the appropriate directory, and the plugin is called to carry out application-specific forwarding actions on this data.

The directory API is implemented by a software component called a directory watcher. The watcher reads outgoing directories and, when it notices new data, calls OCMP to send the data. Similarly, it registers with OCMP to receive data on behalf of the plugins, and, when called by OCMP, writes data into the application-specific directories. Applications can specify configuration parameters, such as the quality of service they want from OCMP, to the directory watcher by writing to a configuration file in their communication directory. Details of this can be found in [2] or in online documentation.

The directory watcher sends and receives data using the OCMP session layer. Essentially, OCMP allows data to be sent and received over one or more connection objects (COs). Each CO wraps a connection, which could be TCP/IP, DTN, or SMS. Unlike socket communication, where the loss of a communication socket is fatal, OCMP 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. A special design feature of OCMP is that it takes application policies into account when allocating bundles to COs. For example, an application may ask some data to be sent with high priority. This could result in OCMP assigning the bundle to an SMS CO. As mentioned earlier, the general problem of allocating bundles to COs is an open problem, and, in current work, we use simple heuristics to allocate bundles to COs.

Because of the clean separation between the OCMP layer and COs, it is possible to easily add new COs to KioskNet. In the current system, we support the DTN, GPRS, and SMS COs. We can, but have not yet, added support for long-range WiFI and WiMAX COs. A WiMAX CO, for example, would allow direct communication between the proxy and a kiosk controller (even with WiMAX, we may still want to use a proxy, so that low-priority bulk data is sent using DTN at low cost).

In addition to the scheduling of bundles over COs, the proxy is also responsible for some DTN routing. Specifically, the proxy has to choose which gateway to send a DTN bundle in order to achieve a performance goal such as delay minimization, or fairness maximization [4]. To do so, the proxy uses the location register (described in Section 3.3) to determine which kiosk a user is currently located. It also implements a scheduler that uses knowledge of bus schedules to make the best possible scheduling decision. Instead of modifying the DTN routing protocol, we decided to make the DTN CO a simple stub that accepts bundles and sends them to a corresponding DTN stub on a gateway. This removes the routing decision from DTN. Also, this means that we do not need to run DTN on the proxy.

Finally, note that SMS and GPRS functionality is provided on the proxy by a recycled cell phone that appears as a serial device to the proxy.

3.1.4 Gateway

The protocol stack at a gateway is relatively simple. Essentially, it is just a standard DTN stack, with the exception of the DTN stub application, which provides one-to-one communication with a DTN CO running at a proxy. All bundles arriving at the DTN protocol stack are given to the DTN stub, which forwards them to the DTN CO at the proxy. Similarly, bundles scheduled to that particular gateway by the scheduler at the proxy are sent by the CO to the DTN stub at the selected gateway. Note that the DTN implementation at the gateway is modified to support selective flooding with meta-data exchange as described in Section 3.2.

3.1.5 Bus

The protocol stack at a bus or ferry is even simpler than at a gateway, because it does not even require a DTN stub. We allow buses to use SMS using a recycled cell phone, though, as of now, we do not have any applications that require or use this functionality.

3.1.6 Kiosk controller

The kiosk controller needs to support shared wired, wireless and SMS connections. Moreover, it serves as a point of shared access for both terminals and mobile devices. Note that, in keeping with its role as an OCMP endpoint, its protocol stack mirrors that on the proxy.

Working our way from the top again (and, for the moment, ignoring the register app and the kiosk agent), we note that the kiosk controller supports several applications, which are

the applications that are used by terminal users, the kiosk franchisee, or by a cell phone. Each such application places data it wants to send or receive in a per-user per-application communication directory, as at the proxy. Again, as at the proxy, a directory watcher serves as an intermediary to carry data to and from OCMP.

We envisage that a kiosk controller is likely to be multiply connected, using both DTN on buses, as well as WiMAX, GPRS, and SMS. Each such mode of connectivity at the kiosk is associated with a CO, and OCMP is used to multiplex bundles to and from the COs as and when they are available. Note that the GPRS and SMS functionality is obtained from a cell phone, typically a recycled cell phone, and the other functionality comes from either the DTN CO running on top of TCP/IP and WiFi, or from a WiMAX CO running on a WiMAX NIC. At the current time, we have not implemented the WiMAX CO.

It should be clear, that this arrangement allows an application running on the kiosk controller to use either a bus, or a fixed link, or SMS to communicate with the proxy, and thence to legacy servers.

We now turn our attention to how the kiosk controller supports terminals and cell phones. Terminals mount communication directories using NFS and run applications locally. Thus, applications running on the terminal simply need to read and write from communication directories using NFS, as shown (we will discuss secure applications shortly). Unlike terminals, cell phones currently do not support NFS. Hence, we need to run the directory watcher application locally on the phones, and, instead of calling OCMP directly, we use a 'kiosk agent' to transfer data from the cell phone to an appropriate directory, when such a connection becomes available. This allows a cell phone to use, for example, the WiMAX connection from a kiosk controller, without having to pay for airtime. Note that all communication from a cell phone to a kiosk controller is over WiFi, which means that our solution is only suitable for smartphones that support WiFi.

We now return to the register application. This application is needed for mobility support. Essentially, when a user is either created at a kiosk, or moves to a new kiosk, we need to tell the location register in the Internet of its new location. We do so by means of the register application, which communicates with a corresponding plugin at the proxy. More details on user registration can be found in Section 3.3.

3.1.7 Cell phone

A cell phone runs one or more applications that communicate using a communication directory, a directory watcher and OCMP. OCMP allows the cell phone to decide whether to use a direct connection, such as over SMS or GPRS, or a connection through a KioskNet. If the KioskNet path is chosen, bundles are sent over the KioskNet CO to a stub called the Kiosk agent running on the kiosk controller. If we wanted to, buses could also run a similar kiosk agent and OCMP, and this would allow cell phones to directly talk to buses.

Note that bundles going through OCMP on the kiosk controller are encapsulated by an OCMP header. To prevent double encapsulation, the KioskNet CO does *not* add an OCMP header. Because OCMP headers are added and removed by COs, this is transparent to the rest of the system. This anomalous treatment of the KioskNet CO reflects that OCMP is an end-to-end protocol. OCMP does not expect to deal with multiple disconnected hops. In this situation, the Haggle protocol suite [7] would probably serve us better.

The current version of the software does not support cellphones using this architecture. Instead, OCMP on the cell phone makes a call to an agent that writes didirectly, and only, to DTN on the kiosk controller. This means that a cell phone cannot use any interface other than DTN on the controller. A more insidious problem is that we have not, as yet, allowed private keys to be installed on cell pphones. So, cell phones cannot receive encrypted data. But this creates a problem at the proxy, which needs to know which users can and which users cannot receive encrypted data. This ought to be in the user profile on DNS, but is currently not supported. Hence, the proxy *always* encrypts data, which means that, for this release, we do not support cell phones.

3.1.8 Terminal

The terminal runs two types of applications: secure applications and insecure applications. Insecure applications use NFS to write to the communication directory on the kiosk controller, as if they were running on the kiosk controller. Secure applications cannot write to the shared directory because unencrypted data written to a directory on the kiosk controller could be snooped on by the kiosk owner. To prevent this, a secure application must encrypt data with the public key of the destination before writing it in the kiosk's communication directory. We use directories, again, to hide the tedious details of security from the application. Instead, the application merely writes data to a 'secure' directory on the terminal's own secure (encrypted) file system (this is also hosted by the kiosk controller, but unreadable by the kiosk owner). Data written to this directory is noticed by a directory watcher, that encrypts the data, and then writes it to the kiosk controller's communication directory. In the reverse direction, when the user logs into the terminal, the directory watcher scans the communication directory on the kiosk controller. Any data found there is decrypted using the user's private key, and placed in its secure directory for consumption by the secure application.

More details of the underlying security protocols can be found in Section 3.4.

3.2 Routing

3.2.1 Naming

As a preamble to the discussion on routing we describe the naming mechanism used in the KioskNet. Note that KioskNet uses name-based forwarding. Therefore, every name is also an address. To avoid confusion, we call them both 'Globally Unique IDentifiers' or GUIDs.

Each user and node in the DTN has a unique user GUID and node GUID respectively. The user GUID is of the form of the form:
username.kioskname.regionname.organizationname.kiosknet.org
and node GUID is of the form
nodename.regionname.organizationname.kiosknet.org . In addition OCMP allows a particular application to be identified by appending the " /application name" suffix to the GUID. We have implemented limited support for wild cards. Any one field in the GUID can be replaced by a "*" character and will match any string. For example

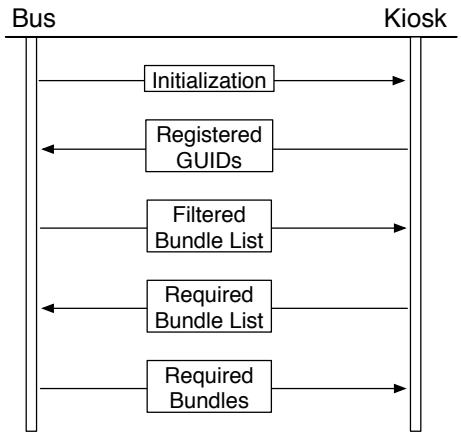

Figure 4: Meta Data Exchange Protocol.

"admin.*.region5.uw.kiosknet.org" will send a bundle to all users called *admin* on any kiosk in region 5 of the uw organization. Similarly
"*.kisok1.region1.uw.kiosknet.org" will send a bundle to all users in kiosk1. This functionality is essential for propagating database updates or software updates. We hope to support more extensive forms of wild cards in future releases.

The KioskNet routing algorithm is responsible for deciding how bundles make their way from a source to a destination, where the source or destination could be a user at a kiosk or a legacy server in the Internet. We consider three cases for routing: routing between kiosks within a single disconnected region, routing to and from a legacy server, and routing between kiosks in different regions. In this section, we assume that users are not mobile; routing in the presence of mobility is described in Section 3.3

3.2.2 Routing within a disconnected region

Measurements show that a ferry can transfer several tens of megabytes of data to and from a kiosk as it passes by, and it can store tens of gigabytes of data on its hard drive. The motivates the use of flood-based routing to trade off over-the-air bandwidth and storage for system reliability and ease of routing.

In naive flooding, a kiosk or a gateway transfers *all* its data to *every* ferry that passes by, and accept data from every ferry. Clearly, this redundancy maximizes the probability of bundle delivery, while eliminating routing decisions altogether. An added benefit is that with flooding, communication between kiosk users in the same region does not require a bundle to go to the proxy. Finally, flooding reduces the amount of configuration at deployment time, making our system easier to deploy.

We avoid the obvious inefficiencies with naive flooding using the following optimization: Kiosks and ferries exchange metadata before data is transferred from a ferry to a kiosk. This lets the ferry know which users are present at a kiosk, and what bundles they have already received. A ferry only transfers bundles not previously received at that kiosk, and for a user currently present there [2]

[2]Note that this optimization prevents the use of a kiosk as an intermediate transfer point between two ferries. We expect this to be unnecessary for most deployments.

The metadata exchange protocol (Figure 4) has five steps. In the first step, the ferry sends a *Hello* message to initialize communication. If the ferry does not receive a properly formated response then it assumes the kiosk does not support meta data exchange and interprets data accordingly. This allows the exchange protocol to be used even if not all kiosks support it. In the second step the kiosk controller transfers a list of user GUIDs registered at the kiosk. The ferry keeps all pending bundles in a series of lists. Each list contains bundles destined to a different destination and the set of lists are stored in a hash-map keyed by the destination GUID. Hence upon receipt of the registered GUIDs the ferry will extract the corresponding lists and concatenate them before transmitting them to the kiosk. This means the filtering of bundles can be done in O(n) time with respect to number of GUIDs. In the fourth step, the kiosk controller determines which of the bundles it does not already have, and requests them from the ferry. In the fifth and final step,the ferry transfers these bundles to the kiosk controller. No metadata exchange is required in the reverse direction: a kiosk transfers all its bundles to every passing ferry. Note that metadata exchange is not carried out between a ferry and a gateway: all gateways accept all bundles.

We now describe intra-region routing in more detail. Consider a bundle with bundle ID B that is to be sent from user $U1$ at kiosk $K11$ to user $U2$ at kiosk $K12$, with both kiosks in region 1. The bundle's destination field is set to $U2.K12$ and kiosk $K11$ gives it to every ferry that passes by. When one of the ferries goes past kiosk $K12$, during the metadata exchange, the kiosk tells the ferry that it has user $U2$, and the ferry tell the kiosk it has a bundle with ID B to the kiosk. The kiosk checks if $U2$ has already received this bundle. If not, the bundle is transferred to the kiosk, and subsequently given to $U2$. If another ferry with B goes past $K12$, during metadata exchange, it will find that $K12$ has already received B, and no further exchange occurs. If, for any reason, the first ferry fails, the second ferry will find that B did not reach $K12$, and can deliver the bundle. This demonstrates how flooding increases reliability. Note that extra copies of B in the system eventually time out and are deleted.

3.2.3 Routing between kiosks and the Internet

We first consider routing in the Internet to kiosk direction. Data from legacy servers destined for kiosk users reaches the user's region's proxy in one of two ways. Either the user tells a legacy server to send ('push') data to the proxy (for instance, by registering the proxy as its mail or instant messenger server), or an application-specific proxy plugin pulls data from the Internet on behalf of the user. This data is then sent to one or more of the gateways to be given to all ferries that go past the gateways.

Proxies are located in bandwidth-rich data centers, but gateways are connected to the Internet typically using slow dial-up or DSL links. Given that the link between the gateways and the proxy is the bottleneck, ideally the proxy should choose only *one* gateway in the region to send each bundle to, rather than flooding it to all the gateways in the region. If the schedules of ferries are known to the proxy, we have developed a routing and scheduling algorithm at the proxy that can choose the best gateway for each bundle and

decide the order in which they are sent in a way that minimizes the overall delay [8]. Moreover, this algorithm can also enforce arbitrary bandwidth allocation among kiosks. If bus schedules are not known (which is the usual case), then the proxy has no choice but to flood it to all the gateways, and this is our currently implemented solution.

We now describe intra-region routing in more detail. Consider a user $U1$ at kiosk $K11$ in region 1, that receives data at its proxy $P1$. When data arrives at the proxy, the OCMP application plugin translates from the application-specific user ID to the user's KioskNet GUID. For instance, a user with email address u@kiosknet.net may be translated to U11.K11.Region1.egov.kiosknet.org. The data is encapsulated into an OCMP bundle B and this destination address is added to the header. The OCMP bundle may be further encapsulated into a DTN bundle, if DTN is used to carry it to the kiosk. If DTN is used, each ferry passing by is given B. Subsequently, the metadata exchange and bundle transfer happen exactly as described in Section 3.2.2.

We now consider routing in the kiosk to Internet direction. When a user at a kiosk wants to send data to the Internet, OCMP creates a bundle with a destination GUID set to the GUID of the region's proxy. The bundle is either sent directly to the proxy on a TCP or SMS connection, or flooded using DTN. If DTN is used, then the bundle is given to all passing ferries, which transfer it to all the gateways within their reach (there is no metadata exchange). The gateway is configured with the IP address of the proxy. On receiving a bundle whose destination GUID is the proxy, it uses a DTN 'always-on' link to give the bundle to the proxy. At the proxy, the OCMP daemon receives the bundle, and passes it to a application-specific plugin for further processing. For example, an OCMP bundle given to an email application plugin would be sent to its eventual destination using SMTP.

Due to flooding, the same bundle may be given multiple times to multiple gateways. To prevent multiple gateways within a region sending the same bundle to the proxy, when a gateway sends a bundle to the proxy, it sends a CLAIM message to all other gateways in the region. When a gateway gets a bundle that has already been claimed, it drops the bundle.

3.2.4 Routing between kiosks in different regions

We allow users at kiosks on one region to send data to users at kiosks in other regions. This is expected to happen rarely, yet may be critical for some applications. To do so, at the sending region, bundles are flooded to reach that region's gateways, which transfer them to the destination region's gateways, where flooding delivers them to the appropriate kiosk. This can be viewed as a composition of the techniques described in the previous two subsections.

More precisely, we now consider the case where user $U1$ at kiosk $K11$ in region 1 wants to send a bundle B to user $U2$ at kiosk $K21$ in region 2. B's destination field is set to the $U2$'s GUID and the bundle is flooded to all the gateways and kiosks in region 1 as described earlier. Because $U2$ is not present at any kiosk in region 1, and this fact is discovered during metadata exchange, the ferry does not transfer any bundles to any of the kiosks in region 1. Ferries do not carry out metadata-exchange with gateways, and so all gateways get all bundles. On receiving B, a receiving gateway uses the bundle's destination field to determine the destination region. As described in Section 3.3, we use DNS to map from the destination region to the set of gateways for that destination region, and from each such gateway to that gateway's IP address. This allows a region 1 gateway, therefore, to transfer the bundle to *all* gateways in region 2, from where it is subsequently flooded (we need to send to all the destination region gateways because the source region gateway does not know which destination region gateway is the best choice to reach a particular kiosk). As before, when a bundle is forwarded, all other gateways are sent a CLAIM message.

Although this scheme is correct, it is inefficient in two ways. First, the source gateway needs to send multiple copies of the same bundle over a bottleneck link. Moreover, it prevents the destination region's proxy from choosing the best gateway to reach a particular kiosk. Therefore, in the next release of our software we plan to have the source region send the bundle to the destination region's proxy, instead of directly to a destination region gateway. We describe this in more detail in Section 3.3.

3.3 Support for mobility

KioskNet users will want to send and receive data from the nearest available kiosk as they move from place to place. This subsection describes how KioskNet supports this functionality.

3.3.1 Globally unique IDs

The key to supporting mobility is to decouple identifiers from locations. Each KioskNet user has a hierarchical location-independent globally unique ID (GUID), as described in Section 3.2.1. Though hierarchical in form, the GUID is location-independent and signifies the home location of the user, that is, the kiosk owner who created the public key for that user. The current location of a user, or, more precisely, the current location from which the user would like to pick up bundles (also called its 'custodian'), is the kiosk identified by a Node GUID (and these are assumed to be static).

3.3.2 DNS-based location register

An Internet-based global DNS-based location register stores translations from each GUID to the GUID of its current kiosk location. This is a generalization of the classical name-to-address translation. The location register also stores, for each region, the set of gateways for this region and their IP addresses. It also stores the IP address of each region's proxy.

When a user is created, or comes to a kiosk that is not its 'home' kiosk, he or she registers with the kiosk. This causes the user's GUID to be added to that kiosk's list of users. Moreover, if this is a new user, or if the user has moved to a new region, the 'register' application on the kiosk controller creates a REGISTER message with the user's GUID and the kiosk's GUID. This REGISTER message is sent to the corresponding plugin on the proxy, which updates the location register with the new kiosk location of the user. This allows the location register to keep track of the current region of every user, albeit with some delay.

3.3.3 Routing to a user who has moved

We consider two cases: a 'near move' where the user moves to another kiosk in the same region (this is expected to be

the common case), and a 'far move' where the user moves to another region (this is expected to be a rare event). Note that we do not support far moves in the current release: this description below is, therefore, a paper design.

Near mobility Near mobility is straightforward: when the user registers with the new kiosk, it becomes part of that kiosk's metadata exchange with a ferry. Thus, bundles destined for that user are received by the new kiosk, and subsequently delivered to that user. Note that if a user moves from one kiosk to another kiosk in the same region, he or she starts receiving bundles immediately. This is because bundles sent to that user are flooded within a region, and a ferry delivers all bundles to the kiosk(s) at which the user is registered. Thus, in-region mobility is seamless. Indeed, because storage is cheap, a user can choose to register at more than one kiosk, and can pick up bundles from any kiosk at which a registration exists: unclaimed bundles are automatically garbage collected when their time-to-live expires.

Far mobility Far mobility is much harder to handle. We consider three sub-cases, depending on whether the source of the data is another kiosk in the same (new) region, a legacy server on the Internet, or a kiosk in another region.

When a user registers with a kiosk in a new region, its GUID is added to the list of registered users at that kiosk. Therefore, it is able to receive bundles from other kiosks in the same region using flooding and metadata exchange, as usual. This handles the first sub-case.

We now consider the case when the source of data is a legacy server on the Internet. In the current version of our software, this data is received by an application plugin at the original proxy associated with that user. Before forwarding it to a gateway, the proxy checks the location register to find the new region of the user (if any) and the set of associated gateways. It then floods a copy of the data to each gateway in that region, because it does not necessarily know which gateway can reach which kiosk[3]. This is correct, but inefficient.

In future versions, we plan to remove this inefficiency by allowing users to switch proxies as described next. When a proxy gets a REGISTER message from a user whose GUID's name hierarchy identifies it as a non-local user, the proxy instructs the proxy for the previous region of that user (which it determines from the location register) to transfer the user's OCMP state to itself. This allows the new region's proxy to fetch and receive data on behalf of the user. This additional processing allows a user who has moved to a distant region to use a nearby proxy, instead of having to use a distant proxy. When a user needs to receive data from the Internet, therefore, it is received by the new proxy. This data is sent using OCMP and DTN to a gateway in the new region, and then flooded, as usual, within the region.

The third sub-case is when a user who has moved to a far region is to receive data from another region. This is not supported in the current version of our system. In a future version, we will support it as follows: When a gateway gets a bundle, instead of always giving it to the local proxy's DTN bundle agent, it looks up the bundle's destination field in the location register to find out whether the bundle should be sent to the region's own proxy, or a distant proxy. If the destination is the region's own proxy, then it forwards the bundle to that proxy, as usual. Otherwise, it uses DNS to find the IP address of the proxy to which it should send a bundle, and sends the bundle to that proxy. Bundles that arrive to a proxy from a remote gateway are handed to the OCMP scheduler for transmission to the best gateway, as decided by the algorithm in [4].

Note that if a user moves from one region to another region, bundles will be sent to the wrong region until the DNS back end is updated. Although this is a problem, we anticipate that its effect will be small in practice because inter-region travel is likely to be rare. Besides, if an SMS CO is available, the interval from the time the user registers its new location to the time that the back end is updated will be small. If neither assumption holds, we need a way to forward wrongly routed bundles. A protocol to do so is described in [2], but we have not implemented this algorithm in our current implementation.

[3]Of course, if a kiosk is also a gateway, then the proxy can send data directly to the kiosk.

3.4 Security architecture

We would like KioskNet to be secure enough to serve as the basis for secure transactions that arise in applications such as rural banking, microfinance, tax and bill payment, and land registry. This requires it to meet the requirements of four distinct groups:

- *KioskNet Franchisers*: Franchisers, usually non-governmental organizations (NGOs) deploying KioskNet, are concerned with the integrity of their KioskNet components (gateways, ferries, kiosk controllers and proxies) and would want to detect, if not prevent, the misuse of their infrastructure by any of the entities named below, including competing franchisers.

- *KioskNet Franchisees*: Franchisees (i.e. kiosk operators) are concerned with the security of their kiosk terminals and would want protection against malware. Franchisers can trust franchisees to issue credentials to users (usually in exchange for a fee), but cannot trust them with user data. In other words, franchisers can create users, but once created, should not be allowed to snoop on user data.

- *KioskNet Users*: Users are concerned with the confidentiality and integrity of their data despite using untrusted ferries and snooping kiosk operators.

- *Application Service Providers*: Depending on the type of service they provide, application service providers (ASPs) would want franchisers to guarantee the integrity of their software when deployed on a KioskNet.

We satisfy these requirements through a combination of standard cryptographic techniques. In particular, we extensively use a Public Key Infrastructure (PKI) to encrypt data and authenticate users. Although well known, PKI has often been thought to be too hard to deploy in the field. In our case, because every KioskNet user and role is a part of the same system, we have a 'closed universe' with a single trusted root certificate authority, i.e. the University of Waterloo. This greatly simplifies the problem. Thus, all the entities named above are issued unique credentials including a 2048-bit RSA private key and a corresponding public key certificate signed by a chain of trust that originates from the University of Waterloo.

3.4.1 Certificates

All the entities named above are issued unique credentials including a 2048-bit RSA private key and a corresponding public key certificate. Public key certificates are issued and signed hierarchically, forming chains in the standard fashion. That is, a secure central root CA server at the University of Waterloo certifies the public key of a trusted franchiser using its own private key. This signature is stored in the form of an X.509 certificate. Franchisers, in turn, issue certificates to their franchisees and ASPs operating in their region. Franchisees automatically certify users registered at their kiosks at the time of user creation. Similarly, all KioskNet infrastructural components, such as gateways and ferries, are issued unique credentials by the franchisers that maintain them. Public key certificates for users, franchisees and ASPs are periodically broadcast throughout a franchiser's region through the use of a public key database maintained at the proxy and replicated at all kiosk controllers. This allows secure messaging amongst the components and users without the need to query a central public key repository, which can be expensive in a disconnected environment. Even with 10,000 users, each with a 2 KByte X.509 certificate, this only takes 20 MBytes, which can be disseminated without too much trouble using KioskNet ferries.

3.4.2 Infrastructure integrity

The security of KioskNet infrastructure is ensured through the use of digital signatures on all remote commands and software updates issued by franchiser administrative personnel. We are mostly concerned with attacks on kiosk controllers, because the devices on ferries and gateways are harder to attack, and, moreover, never see unencrypted user data. To prevent attacks on kiosk controllers, franchisees are not given root access to deployed kiosk controllers, preventing them from modifying the software on these systems. An encrypted root directory at the kiosk controller prevents attackers from removing the device's hard disk and accessing private information off-line (e.g. mounting it on another Linux machine). Industry-standard practices such as the use of intrusion detection systems and firewalls can be additionally used to protect KioskNet components against remote attack through their network interfaces.

3.4.3 Protecting recycled PCs

Recycled PCs (or terminals) are protected against viruses and other malware by forcing them to boot from read-only disk images stored in tamper-evident kiosk controllers. Because only franchiser administrative personnel are permitted to update these disk images, franchisees can be assured of the integrity and security of the operating system and applications running on their kiosks.

The measures taken to protect rural kiosks described above also provide ASPs with assurance of the integrity of the platform their applications are deployed on. Additional security can be provided by ASPs issuing signed certificates for their application binaries, allowing users and franchisees to verify their integrity as required.

3.4.4 User data protection

User data is never stored at a terminal. Instead, it is stored in kiosk controllers and is secured by creating encrypted virtual volumes for each user's home directory keyed with a user-specific *file-system key*. The file-system key is encrypted with the user's password, and this encrypted value (similar to the value in /etc/passwd) is also stored in the kiosk controller's file system.

A user's encrypted volumes are exported over NFS for mounting at kiosk terminals when users login with a valid Unix password at a terminal. Linux's Pluggable Authentication Module (PAM) is used to automate the decryption of these volumes when users login and their encryption when users log out. Users can transparently read and write to their encrypted home directories through our use of the Linux DM-Crypt disk encryption module. Because user data, including private keys, is stored in these encrypted home directories, even attackers with root access are unable to view or modify the data. We emphasize that the user does not need to remember their private key: instead, the user's private key is stored in the user's home directory, and the directory is encrypted with a key derived from the user's Unix password. Thus, the user only needs to remember his or her Unix password. This reduces the cognitive burden on potentially semi-literate users.

In the event that a user forgets his or her password, the file-system key used to encrypt the directory is also encrypted with the franchiser's public key, and safely stored with the franchiser. A user who forgets the account password needs prove his or her identity to the franchiser to recover the file system encryption key. He or she is then given a new Unix account, and the password for this account is then used to re-encrypt the file-system encryption key. This allows for safe recovery from a lost password.

To support privacy for users who are not comfortable using passwords, we envision the use of biometric devices, such as thumbprint readers. We have not, however, incorporated these devices into our system.

3.4.5 Communication privacy and integrity

In-flight user data that requires privacy and authenticity is encrypted and signed at kiosk terminals before it is transferred to the kiosk controller for forwarding to other KioskNet components along its way to the proxy. This ensures end-to-end security of user data, in that this data cannot be read, fabricated or tampered with while in transit within KioskNet.

Note that the traditional approach to ensuring end-to-end secure communication, such as that used in SSL, is to use Public Key encryption to generate a shared secret and use it as a session key for ciphers such as AES. However due to the delay-tolerant nature of the network the time taken by the handshake necessary for generating a shared secret precludes this approach. Using Public Key encryption exclusively is also not feasible as it is computationally expensive for large data sizes. We therefore use AES-CBC with randomly generated 256 bit keys to encrypt data. This key is encrypted using the public key of the recipient and appended to the bundle. Hence recipients can decrypt the data by first decrypting the AES key using their own private keys.

When combined, the security measures described above serve to protect KioskNet against a diverse set of attacks, ranging from simple wireless packet sniffing to more sophisticated attacks that involve removing an KioskNet component's hard disk and booting it with a LiveCD to gain root access and read or modify the data stored in it.

More details of this solution can be found in [9].

3.5 Terminal support

We allow recycled PCs with or without hard drives to boot over local ethernet from a kiosk controller. The recycled PCs are only required to have a BIOS and an ethernet card that supports PXE boot. A recycled PC downloads a Linux kernel from the kiosk controller using PXE and TFTP, and after the kernel is executed, mounts its root file system from the kiosk controller via NFS. To reduce the load on the kiosk controller, it only serves files; all applications are run locally on the recycled PCs. Since all program binaries are read-only, we can guarantee a virus-free environment. Alternatively, if a recycled PC has an operating system installed on its hard drive, a user can elect to boot into that system at boot time. Note that by placing all user files on the kiosk controller, which could even offer RAID storage, we reduce the dependency of our system on the PCs, allowing us to use even marginal hardware.

As mentioned in Section 3.4, to protect user data, each user's home directory is stored in a separate encrypted virtual volume keyed with his/her login password. These virtual volumes are stored on the kiosk controller and exported to kiosk terminals over NFS with the rest of the terminals' root file system. The process of mounting and decrypting volumes when users login, and the reverse when users logout, is automated by the Mount extension to Linux's Pluggable Authentication Module (PAM).

3.6 User management

We allow kiosk owners to perform user management and other system administration tasks through *webmin* [10], a web-based graphical user interface for configuring Unix-like systems. With webmin, kiosk owners can manage their systems without knowing how to use the underlying Linux OS. Webmin also provides a simple interface for kiosk owners to modify their systems, thereby reducing the chance of system failure resulting from human errors.

User credentials (i.e. an RSA private key and corresponding X.509 public key certificate signed by the local Franchiser) are automatically created through an extension to webmin when user is first registered at a kiosk. These credentials are stored in the new user's home directory, which is placed in an encrypted virtual volume, as described earlier in Section 3.4.

Once a user's certificate is issued by the local CA client it must be be propagated to all kiosks. We do so by first updating a central public key database we call the *whitepages* directory. The kiosk controller generates a signed register message containing the user's GUID and X.509 public certificate. This message is transmitted to a gateway using mechanical backhaul and subsequently to the Whitepages server using a TCP connection. The server then verifies the certificate chain and the signature. If the chain and signature are valid then the user's certificate is added to the whitepages directory. It is also possible to update a stale certificate using the same register message or to remove a certificate using an unregister message.

To give all kiosks direct access to the whitepages directory it is replicated on all kiosks. Updates to the central database are periodically broadcast throughout the network to synchronize the copies as described in detail in Section 3.9.4.

3.7 Software installation

	Installation	**Maintenance**
Ofice	Planning, Ordering, Software installation	DTN and sync updates
Field	Physical installation	USB key updates

Table 1: Installation and maintenance tasks for office and field

We have been careful to ensure that the KioskNet system can be installed and deployed with the least possible effort. We assume that the deployment will be conducted by a non-governmental organization (NGO) that has an Internet-connected central office, and that the NGO has a team of field technicians who would do the actual deployment in the field. Accordingly, we divide the installation (and maintenance) process as described in Table 1. To minimize costs, the installation process is designed to take place mostly at the central office. Here, a few trained personnel can carry out the following installation steps on behalf of a large number of kiosks:

1. *Planning:* Deciding the number of kiosks, vehicles, and gateways of the system. We have developed a deployment guide [11] to help NGOs in this process.

2. *Ordering appropriate equipment.* The deployment guide gives detailed instructions on the equipment compatible with our system.

3. *Software installation and configuration:* This step requires loading hard drives with software images from our distribution. We ship our software in the form of a LiveCD DVD A technician can boot any PC from this image into live Linux. The installer then attaches a USB-to-AT2500 IDE connector and an external 2.5" hard drive to the PC. The installation software copies modified Debian Linux images onto the drive through the USB interface. After this copying process, which lasts approximately 12 minutes, the installation software applies user-specified configuration parameters (e.g. IP address, and wireless channel) to the disk image. The disk image is now ready to be deployed in the field. The same process is used to create disk images for kiosk controllers, ferries, gateways, and the proxy.

In the final step, non-expert field personnel physically install the equipment in the villages and ferries. Field technicians do not need to have any knowledge of Linux.

3.8 Maintenance and monitoring

KioskNet needs to be deployed in areas with little or no other infrastructure. Therefore, one of our key design goals was to build a system that could be maintained with the least possible effort by semi-skilled field technicians. We also desired a means to cheaply, securely, and reliably monitor both ferries and kiosks from an NGO central office. These two features would allow a handful of skilled workers at the central office, helped by a larger number of field technicians, to support hundreds or even thousands of kiosks and ferries. In this section, we describe KioskNet maintenance and monitoring.

3.8.1 Maintenance

Routine software maintenance requires software running on kiosk controllers to be upgraded and patched from time to time. To avoid having technicians travel to each kiosk location to install or upgrade software, we provide a sub-system for centralized management and maintenance of kiosk controllers. This mechanism, similar to the Disruption Tolerant Shell [12], is described next.

In KioskNet terminology, an *update* is a zipped and signed file that contains a executable script, the recipients' GUIDs, a unique sequence number, and all other files that the script needs for execution (this is similar to a RedHat RPM). When a KioskNet component receives an update, it first checks the signature. An authentic update is uncompressed in a pre-specified location, and the script is then run with root privilege in a forked shell. When the shell terminates, its sequence number is recorded along with the exit value of the controller script and output logs are submitted to the logging sub-system (described next).

Updates can reach KioskNet nodes over one of three channels. The normal DTN/OCMP mechanical backhaul channel is the preferred transmission mechanism. When this channel does not work, the central office can choose to flood updates to all KioskNet nodes. In rare cases when a node is not reachable using any of these two channels, a field technician can apply the update using a USB key - on detecting an authenticated USB key, the controller reads the update on the key and applies it, just as if it had received it over the wireless link.

3.8.2 Logging

KioskNet has been designed to be robust and tolerant to failures. However, both DTN and OCMP, which are critical software layers, are under active development. Therefore, software failure is a distinct possibility. When a failure does occur, central office technicians require a means to collect and debug system logs that *does not rely on OCMP or DTN*. We have, therefore, designed and implemented a mechanism that floods logs across a disconnected network to the Internet using opportunistic connections. We call this application *log-flood*.

Log-flood periodically compresses the contents of */var/log/*, timestamps it, and signs it with a sequence number. It then periodically sends a broadcast ping to detect neighbouring KioskNet components. When a neighbour is detected they exchange log archives opportunistically using the standard Unix *rsync* utility. For secure transfer, we actually tunnel rsync over ssh using an ssh key installed by the central office when configuring the KioskNet component.

Each KioskNet component floods log archives to each other until the files reach a gateway. To prevent redundant flooding, the gateway does not flood logs to neighbouring ferries; it simply forwards log archives to the proxy on the Internet. The proxy subsequently acknowledges the delivery of each log archive and forwards an acknowledgement file to the gateway. Acknowledgement files are then transferred from the gateway to neighbouring ferries, and flooded back across the disconnected network. When a KioskNet component receives an acknowledgement file, it deletes the originating log archive. Acknowledgement files eventually expire on each component. In this way, by mimicking DTN using rsync, we allow robust log propagation.

3.8.3 Heartbeats

To further monitor the activity of KioskNet deployments each KioskNet gateway, which is always connected to the Internet, sends periodic "heartbeats" to a central server in Waterloo. Heartbeats contain information such as the Linux uptime and the DTN reference implementation statistics. The heartbeat transfer is secured through SSH keys. The heartbeat application checks for updates every time it sends a heartbeat, therefore we can add more fields to the heartbeat later.

3.9 KioskNet Applications

KioskNet applications communicate with legacy servers on the Internet on behalf of disconnected users. Architecturally, applications run on top of the OCMP layer and have two components. The primary component runs on the proxy and a (typically) small helper application runs on the kiosk controller. Applications may be written in any language, including shell scripts. They pass application data to and from the OCMP layer using a directory based API (described next).

3.9.1 Directory API

The Directory API is a branch of the file system where applications place data for OCMP to process [2]. Each KioskNet user has its own directory within the Directory API, and each application has its own branch within each user directory. Each application directory has an upload and download directory to hold data going to and coming from the proxy respectively. These directories hold configuration files that indicate the processing that needs to be done on the files, and can be viewed essentially as a way to pass application-specific parameters to OCMP.

When data arrives in a kiosk download directory or a proxy upload directory, OCMP invokes an application callback specified in the configuration file to handle the newly received application data. We rely on another application, the *directory watcher*, to periodically scan the directories under the Directory API to check for newly added files, which are then passed to OCMP over a loop-back socket. Further details of the Directory API and Directory Watcher can be found in [2]. Although the use of a directory based API is slightly less efficient than passing data to OCMP directly, we found that it greatly eased application integration. In any case, the added delay incurred by polling for updates is negligible compared to the time required to transport data over a mechanical backhaul.

3.9.2 Secure Directory API

Building on the Directory API, the Secure Directory API provides a simple interface for end-to-end secure and authenticated communication. We envision the Secure Directory API being used for a variety of applications, such as bill payment, e-governance, and rural banking.

For every upload and download directory in the Directory API, there is a corresponding "secure upload" and "secure download" directory. A file created in the secure upload directory is encrypted with a nonce using AES 256 with Cipher Block Chaining. The nonce is then encrypted with the recipient's public key. We follow this design because encrypting large amounts of data using RSA encryption is computationally very expensive. Using solely AES encryption is also infeasible as the delay tolerant nature of the network will lead excessive delays in key negotiation.

We have described the procedure to insure secure communication however to support applications such as banking we also require robust mechanisms for authentication and non-repudability. To this end we compute a SHA-1 hash of each secure bundle and then sign it using the senders private key. This signed hash is appended to the secure bundle and is used to authenticate the bundle at the receiving node. This secure authenticated bundles are then output to files in the upload directory of the Directory API. Note that as each bundle is encrypted such that it can only be deciphered by a single user hence if a bundle is addressed to multiple users then multiple copies of the same bundle are written to the directory API upload directory, each copy encrypted for a different user.

The files are then transmitted using the Directory API and appears in the download directory of recipient(s). Any delivered files that are are marked as 'secured' are decrypted and copied to a secure download directory in plain-text form. The secure directories are stored within the user's encrypted home directory. The secure data is also authenticated using the sender's certificate chain and the digital signature contained in each secured bundle.

We have developed several applications that utilize the secure and non-secure Directory API. We now present three representative applications: email, database synchronization, and an opportunistic Flickr[4] client.

3.9.3 Email

The store-and-forward, delay-tolerant nature of SMTP fits perfectly within the KioskNet architecture. Email service within KioskNet consists of five components:

- *Client*: The client application can be any standard email client such as Thunderbird or Outlook Express. The client runs on the recycled PC. The email client is configured to fetch a user's email from the kiosk controller using IMAP. Its outbound SMTP server address is also configured to be the kiosk. From the perspective of the email client and its user, emails are sent and received as if the recycled PC was connected to the Internet.

- *uw-imap*: Any IMAP server can be used to serve emails from the kiosk controller to the email client running on the recycled PC. We chose to use UW-IMAP because it supports the widely-used mbox family of email collection formats, has a small memory footprint, and was simple to deploy compared to other open source IMAP servers.

- *sendmail*: Sendmail implements the SMTP protocol between the email client and kiosk controller and between the proxy and legacy SMTP servers.

- *Kiosk controller component*: The kiosk controller component of KioskNet's email service is implemented as a plugin to sendmail (milter). When receiving emails sent from the recycled PC, the plugin is responsible for intercepting SMTP traffic from the email client and compressing it for transport across to the disconnected network. When receiving an email from a ferry through OCMP, this component translates email from SMTP format into mbox format and adds it to the user's inbox.

- *Proxy component*: The proxy component is also implemented as a sendmail filter. This component receives emails destined for KioskNet users from SMTP servers on the Internet. Like its kiosk controller counterpart, emails are compressed for transport. When handing emails sent from kiosk users, the proxy component simply decompresses the outbound message and passes it to sendmail for delivery.

[4]http://www.flickr.com

3.9.4 Database Synchronization

We anticipate that a major use of KioskNet will be information distribution for data such as agricultural databases and property records. Furthermore, every kiosk must have access to the whitepages directory of public certificates in order to initiate secure communication. Therefore, we wrote *DBSync*, a robust database synchronization mechanism. DBSync periodically takes a snapshot of a central Postgres database using the "pg_dump" utility. The snapshot is distributed to all kiosk controllers using OCMP's Directory API's broadcast facility. Kiosk controllers have a local database and can use the "pg_restore" utility in conjunction with the snapshot to synchronize with the master database. The pg_dump and pg_restore utilities of Postgres lend themselves to this Diff-Patch approach as they use a sequence of insert commands to capture database state as opposed the actual commands applied to the database. Hence the size of the snapshot is independent of the number of changes to the database.

The snapshot size is however dependent on the number of records in the database. Hence this approach is not scalable to large databases. It is also inefficient as a small change in database state will require the transfer of the entire database to all nodes. To ameliorate this we use the Unix *Diff* and *Patch* utilities. All database copies, including the master copy, are initialized to the same blank state and we generate a local snapshot. To synchronize copies with the master we generate a second snapshot at the master, The two snapshots are compared using the Diff utility. The original snapshot is discarded and replaced by the new snapshot and the differences between the two, the update, is propagated to all nodes in the region. Upon receipt of the update a node uses the patch utility to combine the original snapshot with the update. This results in all nodes having the same updated snapshot as the master. We can now use the pg_restore utility to update the database state. This allows us to keep a large database synchronized with minimal overhead as only the changes are have to be transmitted.

There are however several limitations of the approach mentioned above. First, we only allow propagation of updates from the master to local copies. Any changes made to the local copies will be destroyed. Hence DBSync can only be used to propagate databases not collect information from DTN nodes. Second, because we use an eventual consistency model there is no guarantee that at any given point in time all nodes will have the same view of the database. We do not consider this a drawback of our approach as in a delay-tolerant environment such guarantees are meaningless. Finally, as the snapshot consists of a series of insert statements that add each record of the database sequentially there could be significant delay in using the pg_restore utility. Again we do not consider this a drawback as updates

can be performed at night when the kiosk is closed to customers. Despite this we are looking at solutions to this issue one possibility is to use the diff more intelligently. Any new records added to the master database will manifest themselves as insert commands in the new snapshot but not the old one. Similarly any deletions from the master database will show up as an insert command present in the old snapshot but not in the new one. An update is equivalent to a deletion followed by an insertion. Using these observations we can restore the database state by operating on the present data as opposed having to build state from an empty database.

3.9.5 Flickr

This application allows a user at a kiosk to upload pictures to flickr.com. The user takes pictures with his or her WiFi-enabled camera phone. The client-side application on the phone submits the picture files to OCMP using the Directory API. These files are automatically transferred to the kiosk or the ferry, and eventually arrive at the proxy. The proxy-side plug-in for the Flickr application then automatically uploads the pictures to the user's album using the user's credentials through the XML-RPC based API provided by flickr.com. We anticipate that this is useful for NGOs that want to monitor rural infrastructure.

3.10 Wireless Configuration

KioskNet wireless configuration is designed to facilitate access to KioskNet nodes. The kiosk controller in every village is configured as an open access point, therefore all the residents of the village, including NGO workers and other mobile device owners, can associate to it, and take advantage of its services. KioskNet gateways are also configured as open access points.

Ferries in KioskNet serve two roles. When communicating with a kiosk controller or a KioskNet gateway they are wireless stations. The ferries can also act as an ad-hoc node, to receive data from mobile users, possibly bus passengers. The Atheros virtual interface feature, helped us easily implement this configuration.

4. COST STRUCTURE

By design, our solution is low-cost. For instance, we estimate that to provide minimal connectivity to a population of about one million people will require a total capital expenditure of only about $300,000 or 30 cents/person. More extensive coverage will probably cost ten times as much, but still less than a one-time cost of $five per person.

We now present some cost figures. These figures are merely indicative because much depends on the actual deployment environment, and issues such as the rate of interest for small business loans, the import duty rate on electronics, and purchase volumes.

Using off-the-shelf technology, the cost of an average kiosk (which does not require solar power) would be about $450. The main costs at a kiosk are for a single-board computer (such as a Soekris net4501with an 802.11a/b/g mini-PCI Atheros wireless card) which costs about $250, for power remediation (using car batteries), which costs about $100, and for a $100-recycled PC. Note that this cost would be lower with volume purchases. Moreover, the cost of a single-board computer will be lower if local single-board computer manufacturers can be found, or if the single-board computer is replaced with an XO laptop [13]. On the other hand, costs can be higher if there is need for solar cells (which cost around $150), or high-power external antennas, which can add another $250 to the cost.

Assuming an initial capital expenditure of $450, the operational expense, including the cost of field technicians and capital depreciation on an 18-month schedule is about $65/month. The main costs are for a field technician, who can service about 20 kiosks, and the cost of capital depreciation. Even assuming 10% penetration of a target population of 2500 users, with a service charge of $3.00/year, an operator can break even. Additional profit can be generated by charging more per user, by increasing penetration, or by offering additional services, such as computer literacy or digital photographs.

5. PILOT DEPLOYMENT

We deployed a prototype of our solution in Anandapuram, a village in South India, during the week of May 16th, 2006. The bulk of the system was deployed over only two days, which leads us to believe that the system can be rapidly deployed even in environments with little existing infrastructure.

Each kiosk already had a Windows XP PC. We deployed a Soekris net4801 at the kiosk, with a 40 GB Toshiba hard disk drive for local storage. The system was connected to a roof-mounted omnidirectional antenna.

Power came from a 42 AH deep discharge car battery that was charged by two 1200 mA (12V) Powerflex solar panels mounted on the roof of the kiosk. We could also have run our system from AC mains and relied on battery/solar power only for backup.

In the car (see Figure 5), we used power from the car battery, but through an inverter and the Soekris power supply, to mitigate against voltage spikes. The car had a magnetically mounted omnidirectional antenna.

The gateway was in Vishakapatnam. Because the car was parked below the computer room, it was necessary to place the omni antenna outside the building. Figure 5 is a composite figure showing the deployed system.

The purpose of the pilot deployment was to gain confidence in the physical system (antennas, power supplies, single board computers) and their ability to operate with minimal infrastructure and in poor operating conditions – temperatures in the vehicle reached almost 50 celsius! The software infrastructure in the pilot, though, was not well-tested. In the last year, we have thoroughly stress-tested every component of the system, and we released a robust implementation on July 20, 2007.

6. DISCUSSION

Based on our experiences with the prototype deployed in the field [2], we have refined several key architectural components as described below.

6.1 IBE vs. PKI

The initial design of the system provided privacy by means of Hierarchical Identity-Based Encryption (HIBE) [14], an extension to Identity-Based Encryption (IBE). This allows a kiosk user to send authenticated and encrypted messages to another user without the need to know that user's public key. Although useful, using IBE turned out to be problematic

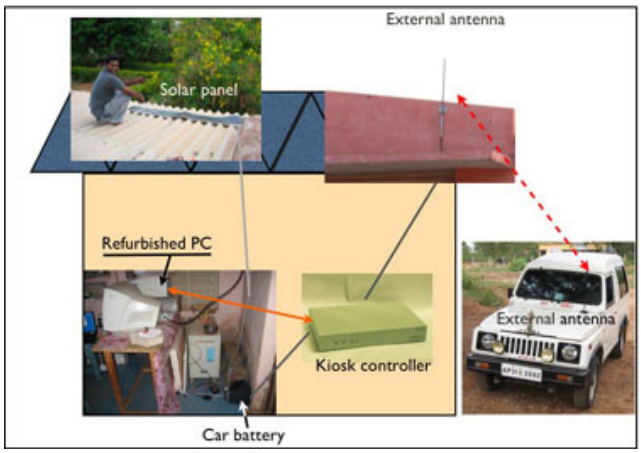

Figure 5: Composite picture of the pilot deployment.

in practice. IBE is essentially controlled by a single entity (Voltage Security, Inc.), which does not release source code and has stringent licensing conditions for commercial use. We therefore decided to replace IBE with our own PKI. There is a wide assortment of open-source tools available for PKI, and we were able to use them to build our own PKI in a matter of a few developer-months.

6.2 Flat names and DHT vs. Hierarchical names and DNS

Our initial design used flat names and a DHT as a Home Location Register to keep track of user location. Again, although this is academically interesting, we found that the DHT we used (OpenDHT) was both slow and unstable. Moreover, OpenDHT is hosted on PlanetLab nodes that are not found in most developing countries. From a technical perspective, a DHT does not allow us to delegate location management for sets of users to third parties. We therefore decided to use hierarchical names for users (of the form user.kioskname.regionname.organizationname.kiosknet.org). This allowed us to use stable, well-tested, and fast off-the-shelf DNS implementations for location management - the location of a user is just an MX record that points to its kiosk. Besides, we can now delegate part of the name space to the organization responsible for a deployment. We think that these two benefits more than compensated for the loss of a flat name space and an infinitely-scaleable DHT.

6.3 Mechanical backhaul vs. Use of all interfaces

When we started our work, we assumed that the only way to reach a kiosk would be using mechanical backhaul. In fact, kiosks are increasingly being reached by SMS/GPRS, and soon, will also likely have WiMAX coverage. Therefore, we decided to support a wide variety of connectivities, with mechanical backhaul reserved for slow and delay-tolerant data. It turns out that using SMS for a control channel brings numerous benefits, such as allowing us to detect ferry failures, and to alert kiosks to turn on their WiFi interface in anticipation of a ferry arrival. We believe that this support of multiple-connectivity makes our system more widely applicable.

7. RELATED WORK

Our work is most closely related to, and was inspired by, the pioneering work by Daknet [3, 15]. However, we differ from Daknet in several ways. To begin with, Daknet focuses only on communication, but KioskNet also supports a computing platform based on recycled PCs. Unlike DakNet, KioskNet leverages DTN for disconnection tolerance, and uses PKI for privacy, confidentiality, and integrity. Moreover, KioskNet supports multiple networks at each kiosk.

The work described here enhances our previously described system for 'mechanical backhaul' described in [2]. Our current system uses different naming, addressing, and routing, as well as PKI-based security as discussed in Section 6.

The goal of low-cost Internet access is shared by the CorDECT project [16] and two well-known long-range wireless projects-Digital Gangetic Plains [17] and WildNet [18]. These are essentially communication technologies and can potentially be integrated into KioskNet as connection objects. In other words, with KioskNet, mechanical backhaul can be used to supplement long-range wireless for delay-insensitive data, such as video content distribution, email, and database updates.

In an alternative use of vehicles, the VIDAL Computer on Wheels project [19] provides a laptop equipped with a CDMA modem in a car that periodically visits villages. Although equally low-cost, this forces villagers to adjust their schedule to that of the vehicle, instead of having their data available to them at a kiosk when they need it.

The use of mechanical backhaul has also been studied in pioneering work on data ferrying [20], and recent work on DieselNet [21]. However, the focus of these projects has primarily been on routing - instead, we take a whole-systems perspective for the specific purpose of rural connectivity.

8. CONCLUSIONS

Rural communities worldwide can benefit from low-cost Internet access. KioskNet attempts to meet this need, focusing not only on the communication path but also many related components, such as security, user management, and log collection. By carefully examining the problem constraints, and integrating well-tested and appropriate existing solutions, we have been able to build a robust system for Internet access without increasing its cost. We look forward to widely deploying it in the field.

9. REFERENCES

[1] K. Toyama, "Review of Research on Rural PC Kiosks," 2007. [Online]. Available: http://research.microsoft.com/research/tem/kiosks/Kiosks%20Research.doc

[2] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Low-cost communication for rural internet kiosks using mechanical backhaul," in *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking.* New York, NY, USA: ACM Press, 2006, pp. 334–345.

[3] "United villages," 2007. [Online]. Available: http://www.unitedvillages.com/

[4] S. Guo and S. Keshav, "Fair and efficient scheduling in data ferrying networks," *Proc. CoNEXT*, 2007.

[5] 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, Dec, 2004.

[6] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya, "A policy-oriented architecture for opportunistic communication on multiple wireless networks," 2006. [Online]. Available: http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/06/ocmp.pdf

[7] J. Scott, P. Hui, J. Crowcroft, and C. Diot, "Haggle: A networking architecture designed around mobile users," *Proceedings of the Third Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, 2006.

[8] S. Guo, "Algorithms and design principles for rural kiosk networks," *University of Waterloo M. Math Thesis*, 2007.

[9] S. U. Rahman, "Kiosknet security," 2007. [Online]. Available: http://blizzard.cs.uwaterloo.ca/tetherless/index.php/Security_Architecture_Overview

[10] "Webmin: Web-based system administration," 2007. [Online]. Available: http://www.webmin.com

[11] "Kiosknet deployment guide," 2007. [Online]. Available: http://blizzard.cs.uwaterloo.ca/tetherless/index.php/Deployment_guide

[12] M. Lukac, L. Girod, and D. Estrin, "Disruption tolerant shell," *Proceedings of the 2006 SIGCOMM workshop on Challenged networks*, pp. 189–196, 2006.

[13] "One laptop per child (olpc)," 2007. [Online]. Available: http://www.laptop.org/

[14] A. Seth and S. Keshav, "Practical Security for Disconnected Nodes," *Proceedings of First Workshop on Secure Network Protocols (NPSEC)*, 2005.

[15] A. Pentland, R. Fletcher, and A. Hasson, "Daknet: rethinking connectivity in developing nations," *Computer*, vol. 37, no. 1, pp. 78–83, 2004. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1260729

[16] A. Jhunjhunwala, B. Ramamurthi, and T. Gonsalves, "The role of technology in telecom expansion in india," *Communications Magazine, IEEE*, vol. 36, no. 11, pp. 88–94, 1998.

[17] "Ruralnet (digital gangetic plains: Dgp) 802.11-based low-cost networking for rural india," 2007. [Online]. Available: http://www.cse.iitk.ac.in/users/braman/dgp.html

[18] R. Patra, S. Nedevschi, S. Surana, A. Sheth, L. Subramanian, and E. Brewer, "WiLDNet: Design and Implementation of High Performance WiFi Based Long Distance Networks," *Proceedings of NSDI*, 2007.

[19] "Vidal computer on wheels project," *http://www.vidal.org.in/node/6*, 2007.

[20] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, pp. 187–198, 2004.

[21] "Umass dieselnet," 2007. [Online]. Available: http://prisms.cs.umass.edu/dome/index.php?page=umassdieselnet