Design and Implementation of the KioskNet System

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

Abstract— **Rural Internet kiosks in developing countries can cost-effectively provide communication and e-governance services to the poorest sections of society. Unfortunately, a variety of technical and non-technical issues have caused most kiosk deployments to be 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 and outline its hardware, software, and security architectures. We describe a pilot deployment, and how we used lessons from this deployment to re-design our initial prototype.**

I. 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 IV 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.
- 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 II and its software architecture in Section III. The security architecture is described in Section III-B. We describe the cost structure in Section IV and our

[1] All figures are in US dollars.

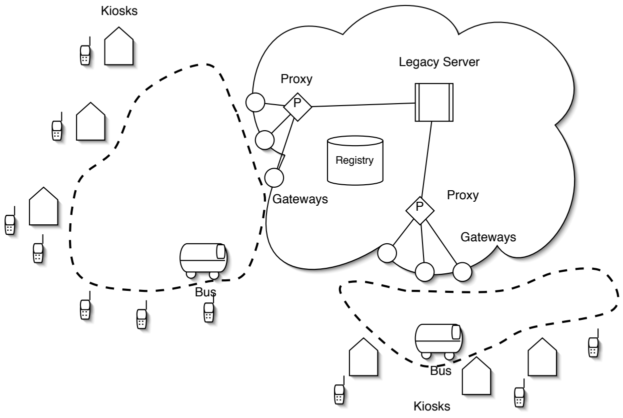


Fig. 1. KioskNet overview.

experience with deploying the system in Section V. Section VI discusses some changes to our initial design decisions that reflect experiences from the pilot deployment. We present related work in Section VII and conclude in Section VIII.

II. 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.

A. 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.

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 be managed by a single administrative entity.

B. 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.

C. 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.

D. 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 III-H.

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. It also maintains a $Whitepages$ database that maps from a user's globally unique identifier (GUID) to its X.509 public key certificate. This database, which is replicated at each kiosk, allows secure communication among KioskNet users.

E. Legacy servers

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

III. Software Architecture

A. Communication architecture

KioskNet communication software runs on proxies, gateways, ferries, kiosk controllers, and cell phones/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. The interested reader will find a more detailed description in [5]: here, we only outline the overall system.

The base communication layer is TCP/IP that runs on wired or wireless network interfaces present at every component in the system. All nodes except for cell phones/PDAs run the Delay-Tolerant Networking (DTN) overlay provided by the DTN reference implementation [6]. DTN provides disconnection-tolerant end-to-end connectivity. We modified the DTNRG's DTN 2.0 reference implementation to add flooding-based routing within each region.

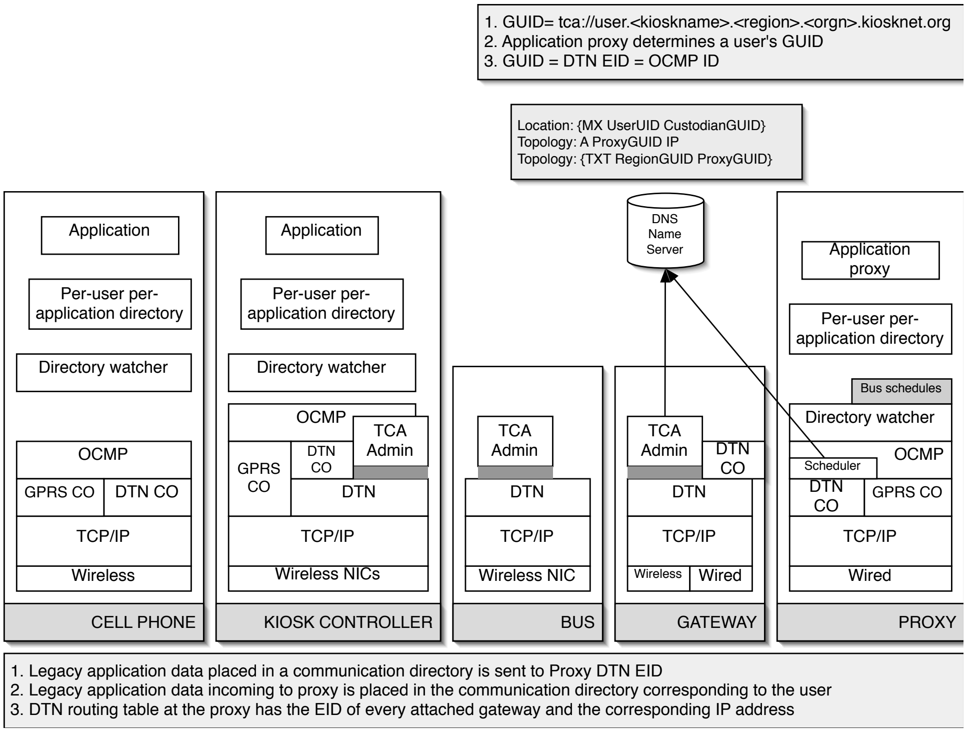
Although DTN provides disconnection-tolerance, it lacks many important services. It does not provide the ability for a kiosk controller, cell phone, or proxy to use application-specific policies to choose from one of many network interfaces, nor does it support mobility for users who may choose to move from one kiosk to another. DTN does not provide application-specific plugins at the proxy. Finally, DTN does not support seamless interconnection with legacy servers on the Internet.

Instead, these capabilities are provided by the opportunistic connection management protocol (OCMP) [2], [7], which runs on top of DTN and other available network connections. OCMP can be viewed as a disconnection-tolerant and policy-driven session layer that runs over both DTN and standard links. Each type of available communication path is modeled as a connection object (CO) within OCMP. For instance, the DTN mechanical backhaul path is encapsulated as a DTN CO. Similar COs can be created 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, as described in more detail in [8]. 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. The design of our system, however, allows the use of more sophisticated scheduling policies without changing the rest of the system. OCMP works in conjunction with the TCA-Admin component, which is responsible for mobility management.

Note that identical Java-based OCMP protocol stacks run on cell phones and kiosk controllers. The only difference is that the DTN protocol stack runs only on kiosk controllers, and not on cell phones. This is because the DTNRG reference implementation, which is written in C++, cannot run on a cell phone. If a Java-based DTN implementation becomes available for cell phones, the cell phone protocol stack can be made the same as on the kiosk controller.

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 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 155 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 slow and limited in message size, it provides a low-bandwidth and low-delay control channel. Additional details can be found in [5].



1. GUID= tca://user.<kioskname>.<region>.<orgn>.kiosknet.org
2. Application proxy determines a user's GUID
3. GUID = DTN EID = OCMP ID

Location: {MX UserUID CustodianGUID}
Topology: A ProxyGUID IP
Topology: {TXT RegionGUID ProxyGUID}

1. Legacy application data placed in a communication directory is sent to Proxy DTN EID
2. Legacy application data incoming to proxy is placed in the communication directory corresponding to the user
3. DTN routing table at the proxy has the EID of every attached gateway and the corresponding IP address

Fig. 2. Software architecture.

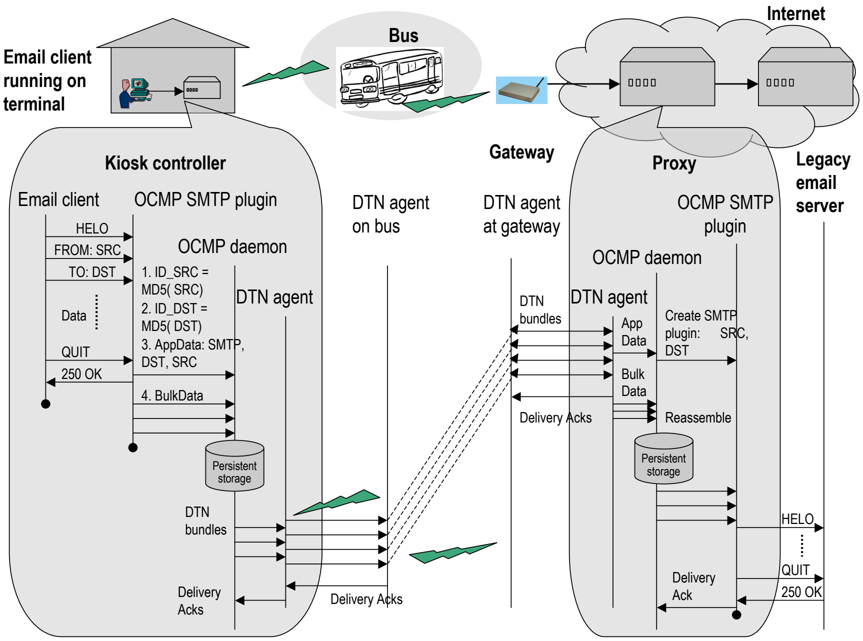


Fig. 3. KioskNet data path when used to send email from a kiosk to the Internet.

B. Security architecture

KioskNet's security architecture is designed to meet the requirements of four distinct groups:

- *KioskNet Franchisers*: Franchisers, usually NGOs deploying KioskNet, are concerned with the integrity of their KioskNet components and would want to detect, if not prevent, the misuse of their infrastructure.
- *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 (ASP) may want franchisers to guarantee the integrity of their software when deployed on a KioskNet.

We satisfy these requirements primarily by (a) using a Public Key Infrastructure (PKI) to encrypt data and authenticate users, by (b) relying on standard Unix security to protect parts of the kiosk controller's file system from the franchisee, and by (c) encrypting user file systems. PKI is often considered to be too hard to deploy in the field. However, 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.

1) Certificates: 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 KB X.509 certificate, this only takes 20 MB, which can be disseminated without too much trouble using KioskNet ferries.

2) Infrastructure integrity: KioskNet infrastructure components are protected through the use of both cryptographic and physical security mechanisms. We are mostly concerned with attacks on kiosk controllers, because devices serving as ferries and gateways are harder to attack, and, moreover, never see unencrypted user data. To prevent attacks on kiosk controllers, franchisees are not given superuser privileges on these devices, preventing them from modifying or introducing any root-owned binaries (particularly the OCMP executable) or configuration files on these systems. Digital signatures on all remote commands and software updates issued by franchiser administrative personnel prevent attackers from using these maintenance mechanisms to remotely modify the software stack or configuration of infrastructure components. The use of locked, tamper-evident enclosures on all infrastructure components prevents attackers from easily removing the devices' hard disks and accessing their private keys offline.

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.

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, $K_{File-system}$. Two copies of each file-system key are maintained: a login key, K_{User}, encrypted with the user's login password stored in the kiosk controller's `/home` directory and a back-up, $K_{User_{back-up}}$, encrypted on the franchiser's public key and stored in `/root`.

Users' encrypted virtual volumes are exported over NFS for mounting on kiosk terminals at the corresponding directory under `/home` when they login with a valid password. Linux's Pluggable Authentication Module (PAM) automatically mounts and prepares these volumes for the on-demand decryption of files when users login, using their passwords to decrypt K_{User} and obtain $K_{File-system}$. PAM reverses the process when users log out. Users can transparently read and write to their encrypted home directories using the Linux DM-Crypt disk encryption module. Because user data, including private keys, is stored in their encrypted virtual volumes, attackers are unable to view or modify their 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.

Users that forget their passwords are required to prove their identities to authorized franchiser personnel, who can then reset their account passwords and use the franchiser's private key to decrypt the corresponding $K_{User_{back-up}}$ keys to create new K_{User} keys. Users would then be able to use their new passwords to login and access their data.

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 secure user data cannot be read, fabricated or tampered with while in transit within KioskNet.

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 components'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].

C. Routing

Routing in KioskNet is based on flooding for maximum reliability. Providing guaranteed delivery of data using mechanical backhaul is hard because ferries are subject to failures and the trajectories of ferries are not always known beforehand. The redundancy provided by flooding maximizes the chance of delivery. Moreover, flooding requires less configuration at deployment time, thereby making our system more rapidly deployable. In KioskNet, flooding is confined in the scope of a region for better scalability.

1) Routing for Upstream Traffic: Bundles sent from a kiosk destined to legacy servers on the Internet are flooded to all reachable gateways in the same region. These gateways coordinate with each other to ensure that each bundle is sent to the proxy by only one gateway to avoid wasting the bandwidth between the gateways and the proxy, which we estimate is the bottleneck of the system.

2) Routing for Downstream Traffic: Data from legacy servers destined to kiosks is first buffered at the proxy responsible for the kiosks. In light of the estimate that the bandwidth between the gateways and the proxy is the bottleneck of the system, the proxy chooses one gateway in the region to send each bundle to, rather than flooding to all the gateways. If the schedules of ferries are known to the proxy, our routing and scheduling algorithm at the proxy can choose gateways for bundles and decide the order in which they are sent in a way that minimizes the overall delay. Moreover, our algorithm can also enforce arbitrary bandwidth allocation among kiosks. After a bundle is sent to a gateway, it is flooded to its destination kiosk.

D. 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 support 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. The kiosk controller 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.

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

TABLE I
INSTALLATION AND MAINTENANCE TASKS FOR OFFICE AND FIELD

E. User management

We allow kiosk owners to perform user management and other system administration tasks through *webmin*, 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 III-B.

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 EID 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. This is described in detail in Section III-H.2.

F. Software installation

Table I summarizes KioskNet installation and maintenance tasks to be done in the central office and the deployment field, respectively. To minimize costs, the installation process is designed to take place mostly centrally. In a central office, few trained personnel can carry out the following installation steps for a large number of kiosks:

1) *Planning:* Deciding the number of kiosks, vehicles, and gateways of the system.
2) *Ordering equipment*
3) *Software installation and configuration:* Loading hard drives with software images from our distribution.

The last step requires a PC and a USB to AT2500 IDE connector. Regardless of the software running on the PC, it will be rebooted, using the KioskNet distribution DVD, into a live Linux session. The installation software copies modified Linux images onto 2.5" hard drives through the USB interface. After this copying process (approximately 12 minutes), the installation software applies user specified configuration parameters (e.g. IP address, and wireless channel) to the disk image.

In the final step, non-expert field personnel physically install the equipment in the villages and ferries. The kiosk in each village consists of a kiosk controller, at least one recycled disk-less PC, rechargeable batteries and solar panels.

G. Maintenance

The need to maintain disconnected and geographically distributed KioskNet components is integral to our design. We have designed a means to cheaply, securely, and reliably, monitor and maintain potentially thousands of components. To remove the need for KioskNet franchisers to travel to each kiosk location (an obvious cost savings), we have designed a sub-system for centralized management and maintenance. KioskNet uses a mechanism similar to the Disruption Tolerant Shell [10] for updating disconnected nodes.

In KioskNet terminology, an *update* is a zipped and signed file that contains a controller script, recipients' DTN EIDs, a unique sequence number, and all other files that the script needs for execution. When a KioskNet node receives an update that matches its EID, it will check the signature. An authentic update is uncompressed in a pre-specified location, and the controller 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.

The controller script performs the following steps:

1) *Checking pre-conditions:* The script may check the sequence number records, along with other preconditions.
2) *Running the main task:* In this stage the controller script can use any of the local files besides the files shipped with the update.
3) *Generating short and long logs:* KioskNet requires that, updates generate two log files. Short logs are immediately reported back to the central administration via the SMS control channel, and long logs are treated as normal system logs.
4) *Returning a status value:* The returned status value is recorded along with the sequence number.

Updates can reach KioskNet nodes over three channels. The DTN/OCMP channel is the preferred transmission mechanism. When this channel does not work we can flood updates to all KioskNet components using rsync during each opportunistic connection between components. In rare cases when a node is not reachable using any of these two channels, a non-technical field staff can apply the update using a USB key.

KioskNet has been designed to be highly robust and tolerant to failures; however, both DTN and OCMP, which are critical software layers, are under active development. As rational system designers, we must conceed that failure could occur. When a failure does occur[2], KioskNet Franchisers require a means to collect and debug system logs. We have designed a mechanism that floods logs across a disconnected network to the Internet using opportunistic connections. We call this application *log-flood*. When enabled, log-flood periodically compresses the contents of */var/log/* and truncates each log file on each KioskNet component. The compressed archive of logs is then timestamped and signed with a sequence number. Log-flood periodically sends a broadcast ping to detect neighbouring KioskNet components. When a neighbour is detected they exchange log archives opportunistically using rsync. For secure transfer, we tunnel rsync over ssh using an ssh key installed by the Franchinser when configuring the KioskNet component.

KioskNet components flood log archives to each other until the files reach a gateway. To prevent redundant flooding, the gateway does not flood logs to neighbouring ferries. The gateway 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.

H. KioskNet Applications

As mentioned in section II-D, applications communicate with legacy servers on the Internet on behalf of disconnected users. Applications are primarily located on the proxy component, with a (typically) small helper application running on the kiosk controller. Application may be written in any language, including shell scripts. Architecturally, applications run on top of the OCMP layer. They pass application data to and from the OCMP layer using a directory based API.

Directory API: The Directory API is a branch of the file system where applications place data for OCMP to process. 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. When data arrives in a kiosk download directory or a proxy upload directory, OCMP invokes a preconfigured application callback to handle the newly received application data. We rely on another application, the *Directory Watcher*, to

[2] We assume the OS remains stable.

periodically scan the Directory API to check for newly added files. Newly detected files are then passed by the Directory Watcher to OCMP over a loop-back socket. Further details of the Directory API and Directory Watcher can be found in [7]. 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. The added delay incurred by polling for updates is negligible compared to the time required to transport data over a mechanical backhaul.

Secure Directory API: Built upon the Directory API, the Secure Directory API provides end-to-end secure communication. For every upload and download directory used in the Directory API there is a corresponding "secure upload" and "secure download" directory. Any files created in the secure upload directory are encrypted, signed and then copied to the upload directory. The sender's certificate is also appended to the data to ensure that only authentic KioskNet users are able to send bundles. The file will be transmitted using the Directory API and will appear in the download directory of recipient(s). Any delivered files that are are marked secured will be decrypted and copied to 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.

While designing the security architecture we faced an interesting problem while choosing the encryption scheme. The traditional approach to ensuring end-to-end secure communication 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 very 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.

We have developed several applications that utilize the secure and non-secure Directory API. We now present three main applications: email, database synchronization, and an opportunistic Flickr[3] client.

1) Email: The store-and-forward, delay-tolerant nature of SMTP fits perfectly within the KioskNet architecture. We expect that email will be the killer application of KioskNet deployments and have designed the application to serve many users. 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 users, 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 purely because it supports the 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*: The kiosk component of KioskNet's email service is implemented as a plugin to sendmail (milter). When handing 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, this component translates email from SMTP format into mbox format and adds it to the user's inbox.
- *Proxy*: 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 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.

2) 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. The snapshot is distributed to all kiosk controllers via the Directory API. When a controller receives a snapshot it applies updates its local database with the snapshot. However, this approach is not scalable since updates have to be broadcast and a single update will require the transmission of the entire database. To ameliorate this problem we generate a *diff* at the central database using the most recent prior snapshot and the current snapshot. The diff is transmitted to kiosk controllers which use the *patch* utility to combine the current snapshot with updates before restoring database state. In this manner we are able to keep large database synchronized with minimal overhead.

3) Flickr: A user can take pictures with his/her WiFi-enabled camera phone. The client-side application will submit the picture files to OCMP using the Directory API. When the user goes past a kiosk or a ferry, the files will be 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.

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

IV. 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 [11]. 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.

V. Pilot Deployment

We deployed a prototype of our solution in Anandapuram, a village in South India, during the week of May 16th, 2006. 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 4), 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 4 is a composite figure showing the deployed system.

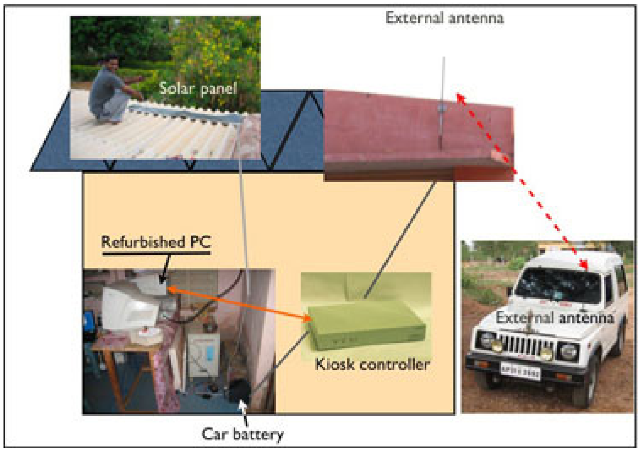


Fig. 4. Composite picture of the pilot deployment.

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 degrees 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. We plan to release the security, SMS, and DNS components of our system in the fourth quarter of 2007.

VI. Discussion

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

A. IBC vs. PKI

The initial design of the system provided privacy by means of Hierarchical Identity Based Cryptography (HIBC) [12]. 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 HIBC turned out to be problematic in practice. HIBC is essentially controlled by a single entity (Voltage Inc), which has stringent licensing conditions for commercial use. We therefore decided to replace HIBC 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.

B. 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.

C. Mechanical backhaul vs. Use of all interfaces

We initially assumed that the only way to reach a kiosk would be using mechanical backhaul. In fact, kiosks are increasingly being reached by GPRS, and soon, will also 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 acceptance of multiple-connectivity makes our system more widely applicable.

VII. Related Work

Our work is most closely related to, and was inspired by, the pioneering work by Daknet [3], [13]. 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 VI.

The goal of low-cost Internet access is shared by the CorDECT project [14] and two well-known long-range wireless projects- Digital Gangetic Plains [15] and WildNet [16]. These are essentially communication technologies and can potentially be integrated into KioskNet as connection objects.

In an alternative use of vehicles, the VIDAL Computer on Wheels project [17] 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 [18], and recent work on DieselNet [19]. 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.

VIII. 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.

References

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

[2] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, "Low-cost communication for rural internet kiosks using mechanical backhaul," in *Proc. ACM MOBICOM*, 2006.

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

[4] S. Guo and S. Keshav, "Fair and Efficient Scheduling in Data Ferrying Networks," *Proc. CoNEXT*, 2007.

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

[6] 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.

[7] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya, "A Policy-Oriented Architecture for Opportunistic Communication on Multiple Wireless Networks," *http://tinyurl.com/2j9ewt*, 2006.

[8] M. Zaharia and S. Keshav, "Fast and Optimal Scheduling Over Multiple Network Interfaces," *University of Waterloo Technical Report, CS-2007-36*, 2007.

[9] S. Ur Rahman, U. Hengartner, U. Ismail, and S. Keshav, "Securing kiosknet: A systems approach," *University of Waterloo Technical Report, CS-2007-43*, 2007.

[10] M. Lukac, L. Girod, and D. Estrin, "Disruption tolerant shell," *Proceedings of the 2006 SIGCOMM CHANTS*, pp. 189–196, 2006.

[11] "One laptop per child," *http://www.laptop.org/*, 2007.

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

[13] 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

[14] 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.

[15] "Ruralnet: Low-cost networking for rural india," *http://www.cse.iitk.ac.in/users/braman/dgp.html*, 2007.

[16] 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.

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

[18] W. Zhao, M. Ammar, and E. Zegura, "A message ferrying approach for data delivery in sparse mobile ad hoc networks," *Proc. ACM MOBIHOC*, 2004.

[19] "University of massachusetts dieselnet project," *http://tinyurl.com/37kcfg*, 2007.