A Postal System Based Digital Network And A Distance Learning Application

Nitin Garg

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

September 2006

Abstract

In this thesis, we propose the novel approach of turning storage media transported by the postal system into a general-purpose and transparent digital network, extending pervasive, high-bandwidth, and low-cost connectivity to places such as rural areas in developing countries. We call such a system the *Postmanet*. To fully realize its potential, however, an end user needs better support than being told to burn discs and toss them into the mail bin. We describe the systems support that we provide in order to achieve the generality, transparency, efficiency, and scalability goals of the system. The issues that we address include: managing "DVD robots" that automate mass-processing of DVDs, application-specific marshaling and unmarshaling of messages, providing best-effort reliable and secure delivery, simultaneous exploitation of conventional connectivity, and a mechanism for distributing and updating application code. Two additional support features are of particular importance.

The first is a distributed object repository that makes available a single name space, on which any sites, including those that lack conventional networking access, can perform read, write, navigation, search, and other operations. This high-level abstraction makes it easier to construct distributed Postmanet applications. It also helps us realize a powerful "network effect," as spontaneous connections are established among sites that enjoy shared access to a common repository. The second is scalable routing. Simply leaving end users to directly swap discs with each other does not scale well, because as many as N^2 discs may need to be exchanged at once in an N-node network. We solve this problem by multiplexing/de-multiplexing data to/from a smaller number of discs in transit. This can occur multiple times at dedicated nodes inside the network, or at peer end user nodes. We present routing topologies that can result in a good balance between simultaneously minimizing the number of discs involved and the end-to-end postal latency.

We have built and deployed a real-world application, a rural distance learning system called the Digital StudyHall, on top of the Postmanet. It consists of a network of hubs and spokes, where the hubs are urban centers of excellence, which “radiate” content and methodology into poor villages and slum schools. Our experiences in rural India not only have provided us insights on the type of the systems support that we need, but also have allowed us to study mediation-based pedagogy that has proved promising in extending high-quality education to a needy population. For more information, please visit: http://dsh.cs.washington.edu.

Acknowledgments

The completion of this thesis marks the end of a long and wonderful journey for me. As I look back on my life as a student, I feel a tremendous sense of gratitude and satisfaction. I am thankful for the fortunate turn of events throughout my life that finally brought me to Princeton nearly six years ago. These last six years have been absolutely amazing. Almost everyone that I met and interacted have been really helpful, nice and very interesting people to know. My education has been more than in just computer science and for that I am and will eternally be grateful. I would like to take this opportunity to thank some of these individuals in particular.

Sumeet Sobti, has been a great friend and colleague. He helped me adjust to my new life as a graduate student in a new country. He was always fun to work with and had insightful thoughts and interesting perspectives on any problems we encountered. I thank Sumeet for his patience, his support and his friendship.

Abhra Mitra, was always a wonderful friend and roommate. It was an absolute delight discussing just about any topic with him. His wit, insights and creativity made any discussion sparkle. I will always cherish and miss the long conversations we had over evening tea and while cooking dinner.

The best thing to happen to me on a personal level was meeting Anindita. She is my counsel, my conscience and my pillar of support. I am glad I met her, I am glad I came to know her and I am glad that she is now my better half. Life would simply not have been the same without her.

I am thankful that I found a wonderful advisor in Prof. Randolph Wang. Randy has been more than just an advisor to me – he is also a great friend. When it comes to work, he is tough, demanding and has a tremendously insightful and incisive mind. His creativity and boundless energy and enthusiasm is very inspiring and he has always pushed me to do my best. Combined with an otherwise good natured, easy going and cheerful outlook to life, he has been a absolutely wonderful teacher. I have learnt a lot from him and have grown not just as a researcher but also as a well rounded person.

Another source of my learning and growth as a researcher has been working with Prof. Arvind Krishnamurthy. Arvind's brilliant insights and wide range and depth of knowledge made it a pleasure to work with, and learn from him.

Last but not the least, I would like to thank all those that have made graduate school a very pleasant learning experience. I thank my teachers that I had the privilege of learning from as their TA, especially Prof. Brian Kernighan, Prof. Kai Li, and Prof. Jaswinder Pal Singh. I would like to thank the department staff, especially Melissa Lawson and Jennifer Widdis, who are always nice, helpful and patient. I thank all my office mates from over the years in the department, especially Junwen Lai, who I could rely on for fun conversations and weekly helpings of Newsweek magazine.

I am grateful for all the help and support I received without which this thesis would not have been possible. My work was supported by NSF grants CCR-9984790 and CCR-0313089 and by Microsoft Research India. I thank Kentaro Toyama of MSRI for his unwavering support of the project. I thank all the teachers and staff of the StudyHall school. Without their help, our stay in Lucknow would not have been as pleasant. I especially thank Dr. Urvashi Sahni, principal of the StudyHall school, for allowing us access to her school, for her enthusiastic participation in the project, for being a leader of the Digital StudyHall, and for making us feel at home in an unfamiliar city. The Digital StudyHall would not have been possible without her. I also thank Raghuvansh Kapur for his enthusiastic help and participation in setting up the Digital StudyHall.

Finally, I would like to thank my committee for their help and support in culminating my life as a graduate student. I thank Prof. Li and Prof. Cook for taking time out of their busy schedules to read and give valuable feedback on my thesis. I thank Prof. Martonosi and Prof. Singh for being part of my committee and giving me a chance to present and defend my work.

As I look back with fondness on these six years past, I also look forward with excitement at the road that lays ahead. I feel glad that the day has come when I finally get to write down these words.

Dedicated to Mimi Garg.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the adoption rates of the Internet and broadband connections slow down in the U.S., latest studies [39, 26] suggest that the “digital divide” could be solidifying: those with modest incomes, rural residents, and minorities are among those who lag behind in Internet access. Most people living in developing regions, who represent an overwhelming majority of the world’s population, also largely fall on the “wrong” side of the digital divide.

Bridging this digital divide, especially by attempting to increase the accessibility of “conventional” broadband connectivity, can be challenging. The improvement of wide-area Internet bandwidth is constrained by factors such as how quickly we can dig ditches to bury fibers in the ground. The cost of furnishing “last-mile” wiring can be prohibitively high, and the progress has been excruciatingly slow. Satellite-based solutions have severe cost and aggregate bandwidth limitations. The history of all previous revolutions in “communications technology” (such as the railways, and the telegraph) bears witness to the fact that their growth, deployment and increased usage takes place over many years. Moreover, their deployment is dictated by economic forces that result in poor or rural areas significantly lagging behind. Government action, via tax incentives or policy enactments, can provide the impetus to providers to invest in building the infrastructure but this too is limited in poor and rural areas.

1.1 The Postmanet

It has been noted that by mailing a stack of DVDs via regular postal mail, one can get greater effective bandwidth than is possible by currently available broadband solutions. Hence, instead of waiting for the uncertain takeoff of a number of existing and proposed technologies, which can be many years away, we propose [67, 68, 50, 16] to turn the *existing* world-wide postal systems into a generic digital communication mechanism as digital storage media is transported through the postal "network." The proposed system is dubbed the *Postmanet*.

The idea of sending digital content via the postal system is not new. Companies such as AOL and Netflix have used this approach to deliver software and movies on a large scale, and some researchers have reported shipping hard disks filled with astronomy data [23]. None of these existing attempts, however, have turned the postal system into a *generic* communication channel that can cater to a wide array of applications. Let us take a closer look at what advantages such a system would offer, and why we would want to build such a system.

1.1.1 Postmanet Advantages

Compared to more conventional wide-area connectivity technologies, the Postmanet enjoys several important advantages.

• *Wide reach.* The postal system is a truly global "network" that reaches a far greater percentage of the world's human population. For some parts of the world, it is about the only "network" that covers them.

• *Great bandwidth potential.* While the bandwidth potential of a "sneaker net" is well known, some may consider it to be a temporary fluke stemming from the relatively poor capacity of today's Internet. We, however, believe that this is not necessarily the case, if we examine some fundamental technology trends. Storage density of flash memory and magnetic disks has been increasing at the annual rate between 60% and 100%, and it is likely to continue in the foreseeable future. This tremendous rate of improvement is likely to be almost directly translatable to the amount of bytes transportable by the postal system for a fixed cost or in a fixed volume. Besides flash memory and hard disks, the next generation Blu-Ray DVDs can hold up to 27 GB per disc today. Sony is planning to commercialize a 4-layer 100 GB version in 2007, and it has already successfully developed an 8-layer 200 GB version [38]. Hitachi Research has recently announced multi-layer technologies that can produce 150 GB discs by 2007 and 1 TB discs shortly thereafter [57]. One can also ship multiple units of these storage devices. As better storage devices become available, they can be instantaneously and incrementally translated into Postmanet bandwidth improvements.

In contrast, the wide-area network bandwidth growth is often constrained by labor-intensive and costly factors such as how quickly we can dig ditches to bury fibers in the ground, how quickly we can furnish last-mile wiring to homes (an endeavor that can be prohibitively expensive), how quickly we can launch satellites, or how quickly we can erect WiMax (the longer-distance versions of WiFi) towers. These factors are unlikely to improve faster than the exponential growth rate of storage density. Satellite- and WiMax-based solutions may face aggregate bandwidth limitations. And the future of some of these alternatives (such as WiMax) is far from certain. Far from being a temporary fluke, the bandwidth gap between Postmanet and more conventional alternatives is likely here to stay and, indeed, widen. We do not, however, necessarily view Postmanet as a competitor to these other alternatives. Before better alternatives become a widely deployed reality, exploring the Postmanet, an alternative that can already deliver practically infinite bandwidth today, may foster the development of and demand for sophisticated bandwidth-intensive applications, which may one day readily migrate onto alternative connectivity technologies.

• *Leveraging existing infrastructure.* It is no fluke that the DSL and cable modems have emerged as the most popular means of broadband connectivity in the US. Being able to leverage an existing infrastructure has certain advantages over having to start from scratch. First, the marginal cost of upgrading an existing infrastructure so as to be able to provide a new service over it, more often than not, is cheaper than the cost involved in building a new infrastructure. Second, existing services over the existing infrastructure help sustain the provider through the process of slowly adapting, upgrading or replacing the infrastructure to be able to support the new services better. For example, the early cable systems could only carry signals in one direction: from the cable operator to the subscriber. This was due to the presence of "signal amplifiers" at regular distances. In order to be able to provide Internet access, cable operators had to replace all the old amplifiers with new ones which allow signals to travel in both directions. They also had to install new equipment in their centers to handle Internet traffic. Subscribers that sign up for the service have to be provided with cable modems. All of these changes are expensive, and take time; but a competitor starting from scratch has to invest in building the network itself, building centers, building trained staff etc., which is far more costly than the costs incurred by the cable operators. In the poor "third world" regions, however, the telecom and cable industries either do not exist or are still in their infancy and largely serve only the urban areas. In the remote and rural areas, they have no existing infrastructure to leverage and unlikely to be able to provide broadband access on a large scale for quite some time to come.

The postal system has had over 200 years of head start in building its network. To leverage the postal system for digital communication, one needs no significant new investment in exotic equipment. While it would be impossible to provide DSL service without the involvement of the telecom operator, it is possible, albeit not entirely desirable, to build the Postmanet without the existing postal system making any changes or even being aware that it is offering the service. This is, in fact, the case with Netflix, which does not rely on any extra postal system cooperation to run its service. This frees users from having to wait for the postal service to get its act together and make the service available. Any enterprising group wishing to become a "Postmanet Service Provider" can step in to provide the service as it is not held hostage by the need for the involvement or co-operation of the postal system. Indeed, individuals desiring a Postamanet-like service can take the matter into their own hands in a peer-to-peer system, instead of waiting for *any* service provider.

• *Low cost.* According to recent studies [36], about a third of non-Internet users cite cost as their reason for eschewing the Internet. The low cost advantage of the Postmanet should be attractive to average households, content offerers, and "power users" alike. The goal of providing citizens with affordable access to postal service is typically an integral part of most nations' postal system charters. In the U.S., even if each household sends (and receives) one DVD each day, the monthly cost of about $12 compares favorably with existing ISP offerings, especially if we were to consider its vast bandwidth potential. The relatively liberal use of the postal system by AOL and Netflix highlights the low cost advantage of this approach to content offerers. The availability of a public transit system like the Postmanet infrastructure, which allows each household to receive (per postman visit) a single disk that contains customized content from multiple content offerers, can further reduce the cost to all involved. In addition to catering to "low end" users, the cost advantage of the postal system relative to that of a high-speed wide-area network also holds for corporate "power users" shipping large amounts of data [23].

• *Good scalability.* The postal system appears to have tried and tested experience dealing with "flash crowds" such as those seen on tax days or certain holidays.

• *Ease of incremental adoption.* A single pair of Postmanet users can already derive useful value from the system, without having to wait for a massive-scale user community or world-wide infrastructure to develop. From this modest start, the system can grow gradually. This incremental deployment may circumvent the classic "chicken-and-egg" problem associated with the difficulty of simultaneously developing infrastructures, applications, and user populations.

1.1.2 Goals

The goal of exploring the Postmanet approach is to extend, to complement, and/or to even foster other alternatives, instead of competing against existing or future alternative network access modes.

• *Extending the Internet.* For those who have no access to connectivity or access to only low-bandwidth connectivity, the Postmanet can provide an inexpensive connectivity alternative to enable certain networked applications, especially bandwidth-intensive ones. Most people in most of the developed world have access to at least a telephone line through which they could connect to the outside world. The vast majority of humanity, however, lives in the under-developed, impoverished regions of the world (such as parts of western China, rural India, and sub-Saharan Africa). The Postmanet can connect these regions to the rest of the "connected" world and more importantly, provide connectivity within these regions.

• *Complementing the Internet.* The Postmanet has long (but reasonably predictable) latencies. A single DVD disc mailed via first-class mail takes at most 3 days to get to its destination within the continental US. Each such DVD "packet" can also deliver over 4GB of data. We call such a channel a High Latency High Bandwidth (HLHB) channel. Correspondingly, we call a traditional Internet connection a Low Latency Low Bandwidth (LLLB) channel. For places that have access to both an HLHB channel and an LLLB channel, an interesting problem is how to exploit an integrated and simultaneous use of *both* channels to get the best of both worlds. For example, small requests, acknowledgments, "NAKs," and control messages may be sent along the LLLB Internet, while large messages are staged on mobile storage devices for transmission by the HLHB postal system. Another example of the complementary nature of the Postmanet is that it may increase the availability of the communication subsystem: if the Internet is down for some reason, one still has another alternative.

• *Fostering application development.* The Postmanet is likely to be more quickly realizable compared to more ambitious efforts of making high-bandwidth connectivity widely available. Bandwidth-intensive applications developed for the Postmanet, users who become accustomed to its benefits, and lessons learned can potentially be transferred to the alternatives farther away on the horizon when they become real.

1.1.3 Challenges

At first glance, manual preparation of data for shipment on movable storage media may appear deceptively simple. But manually copying, naming and managing many messages, potentially for numerous applications and communicating peers, is cumbersome. The postal system represents a classic analogy of a datagram service: individual movable storage media may be damaged, lost, delayed, or delivered out of order. Human users or individual applications should not have to cope with these complications if they desire better guarantees and abstractions. What makes these issues even more complex is our desire to simultaneously exploit the Internet and to exploit the excess capacity of movable storage media to improve the latency, cost and reliability of the system. The key challenge, in other words, is to build the Postmanet to be *generic*, *transparent*, *efficient* and *scalable*. The efficiency and scalability requirements are self explanatory but we elaborate on what we mean by the other two.

By *generality*, we mean that the Postmanet should be able to cater to a variety of applications without each application having to come up with their own Postmanet-like mechanism. The provision of an application-neutral Postmanet "public transit" system that is easily and cheaply exploitable by any potential communicating parties is important. This is analogous to the fact that the existing Internet is such a generic infrastructure. Without it, a potential innovator who is interested in developing a Netflix-like application may need to reinvent the whole infrastructure from scratch. The coexistence of multiple Netflix-like infrastructures can lead to various forms of inefficiency. Smaller players may not be able to afford to put up their own infrastructure at all.

By *transparency*, we mean that the Postmanet should minimize the need for manual handling of the storage media being transported and hide the details as much as possible from the users and applications. The Postmanet users or applications need not manually inspect or process the content of the received storage media; the users or the applications need not manually stage or copy data; and the users or the applications need not worry about issues such as potential loss or damage of the storage media in the postal system. Unlike an AOL or Netflix user, who must know what application-specific steps to take to manually handle the occurrence of these events, a Postmanet user's direct involvement in the running of the Postmanet system should be, at most, limited to mailing the outgoing storage media and to plug in the received storage media. This is analogous to the fact that low-level details such as packets and routers are minimally visible to a conventional network user. We emphasize here that we do not consider transparency to mean making Postmanet itself "transparent" to *existing* applications. In other words, we do not consider it our mission to enable existing applications, running over the existing conventional networks, to run over the Postmanet *unmodified*. Rather, we want to provide a "transparent" interface that would allow developers to port their applications to run over the Postmanet.

1.2 An Example Application: A Distance-Learning System

The difficulties and the advantages of our approach are best illustrated if we consider a real-world application that many people have tried to tackle, but with great difficulty and limited success in developing regions. Providing access to affordable high-quality primary education is an important tool in the fight against poverty, especially in the third world countries. Distance learning or "e-learning" aims to leverage and extend existing educational resources through judicious deployment of information and communication technology (ICT). It is important to note that our goal is not to compete against or replace human teachers; on the contrary, our goal is to amplify the reach and power of the limited number of qualified teachers that are available. The e-learning landscape is littered with misguided and expensive "wire-the-schools" projects that have little to show for in the end. We believe that one of the reason that contributed to some of the previous failures is the insufficient attention given to evaluating the *"cost realism"* of the projects. Consider the case of India: a recent survey of schools in the state of Bihar, Madhya Pradesh, Uttar Pradesh, and Rajasthan [34], shows that 63% of the schools have no functional roofs, 58% have no drinking water, 89% have no functioning toilet, 27% have no blackboards, and 8% have none of the above! The high cost of a large-scale conventional "wire-the-schools" attempt must be carefully weighed against these pressing basic needs. Priority-setting and cost/benefit analysis is crucial [41]. Keeping the per-student expenditure down would allow our system to eventually scale up to encompass a greater number of schools and students. The Postmanet provides a realistic, cost-effective connectivity mechanism for rural and remote areas and can enable distance learning systems to be built to serve these regions.

We have built such a distance learning system, called the "Digital StudyHall," which aims to "connect" resource-starved schools in rural India to well-equipped urban schools so that they may benefit from the better human and content resources available in the urban environments. A test-deployment of the Digital StudyHall has been set up and under active use in Lucknow, in the state of Uttar Pradesh, India since June 2005. It connects "StudyHall," a highly regarded school in the city of Lucknow to two village schools and one school for girls from urban slums. The Digital StudyHall, or DSH, consists of several components[1], one of which is the "Learning eBay." The Learning eBay or the "Repository" is a site that connects learners and teaching staff across time and space; it is analogous to how an auction site matches supplies and demands. Volunteers and professionals all over the world may contribute content, monitor remote mediation-based classroom sessions; communicate with each other, other teachers, and students in discussions on various topics such as pedagogy research. Students and staff with limited experience or resources may tap into this site to augment their learning. Such a site would allow volunteers (potentially from overseas) to make flexible time and location commitments. The site is headquartered at the StudyHall school in Lucknow in our pilot deployment. People all over the world with conventional network access can access the site via the web. People who have limited or no conventional network access, such as those in the village schools that the Digital StudyHall serves, access the site via the Postmanet.

We discuss more details of the Digital StudyHall in Chapter 3. The Digital StudyHall figures prominently in this thesis because: (1) it demonstrates the kind of compelling and exciting applications that the Postmanet approach enables; and (2) our preliminary experience with this application in the real world has allowed us to gain significant insight into the type of systems support challenges that we must address to fully realize the potential of the Postmanet approach. We discuss these issues in the following subsections.

[1]The other major components of DSH include EdTV—a mechanism that uses cheap TVs as "networked thin client displays" to address the "display problem" in village schools: the problem of not having enough computer displays for all students to see clearly; and a platform for conducting mediation-based pedagogy research. More details can be found in [66].

1.3 Postmanet-enabled HTTP

Web browsers are becoming increasingly popular application development platforms, especially for services accessible or sharable over networks. A browser-based user experience is also one that non-sophisticated users (including even the rural audience that we target in our distance learning application) have been exposed to or can get accustomed to quickly. Browser-based usage experiences, despite their deceptively simple interfaces, can provide rich functionalities effected by sophisticated server-side scripts. Once we "crack the code" of making a browser-based interface work on top of the Postmanet by providing an application-neutral API, we might be able to more easily enable a rich set of additional applications (such as email and file sharing) with our Postmanet-based approach. For these reasons, we have made a conscious decision of engineering the Digital StudyHall application with a browser-based interface, which, in turn, is designed to work on top of an application-neutral Postmanet API. In addition to the ease of use and the potential payoff of generalizing to other applications, an additional reason for making the Digital StudyHall application accessible through a browser-based interface is that such an interface provides a natural way of integrating the Postmanet-based communication and traditional web-based communication, because the Digital StudyHall repository is meant to be accessible via both the postal system and the conventional Internet.

The considerations described above are from the point of view of user experiences. From the point of view of implementation ease, it is also natural to first consider a browser-based interface for accessing Postmanet-enabled services. The http protocol provides well-defined request/reply communication points; and there are well-established precedents of using techniques (such as mirroring, caching, and offline browsing) to deal with traditional communication difficulties (such as disconnection, poor bandwidth, or mobility). Such precedents are a natural inspiration for us to consider when we begin to design an asynchronous communication subsystem built on top of the Postmanet.

Despite the seeming simplicity of browser-based interfaces and the existence of precedent techniques of supporting limited operations during communication failures, building an application-neutral Postmanet layer is not as simple as it may appear at first. "Marshaling" and "unmarshaling" request and response messages onto and from movable storage media is only one of the most obvious responsibilities of such a layer. Providing a coherent and application-neutral client cache that is refreshed by the Postmanet is another example responsibility of this layer. Making such functionalities work is much more complicated than what traditional mirroring or offline-browsing techniques can support.

An offline browser relies on a traditional communication link to be restored at some point in time; while a Postmanet-supported browser, in the absence of a complementing LLLB link, is never connected via a conventional network. Furthermore, during disconnections, offline browsing has no provision of supporting interaction with server-side scripts. This is a handicap that a Postmanet-enabled browser application must address. In our approach, we address it by explicitly migrating some code (as well as data) from the "server" to the "clients" so the client browsers continue to function in absence of any conventional connectivity. As code is updated on the server side over time, there also needs to be a systematic mechanism of transmitting code updates to the client side. Furthermore, in a traditional mirroring system, only data updates at the server are transmitted to their mirrors. In our system, we must allow each client to contribute data back to a repository. As one can see, the level of systems support that we need to enable a Postmanet-enabled browser application is considerably more sophisticated than what traditional caching, mirroring, and offline-browsing techniques can provide.

The desired level of systems support does not stop at the software issues described above; there are also mechanical and hardware considerations. A traditional web server handles incoming/outgoing messages from/to many web clients. In a similar vein, a Postmanet-enabled server should be able to handle incoming/outgoing movable storage media (such as DVDs) sent from/destined to many Postmanet-enabled client browsers. The question we pose is how one should handle the mechanical aspects of DVD processing. A naive answer is to have a human operator sit next to the server, and the operator's sole job is inserting/removing DVDs into/from DVD drives all day. This naive approach is obviously tedious, slow, expensive and does not scale well. Our approach is to employ off-the-shelf "DVD publishers" (or "DVD robots"). The originally intended purpose of these DVD robots was mass-duplication (for mass-advertising campaigns, for example). We have adapted these DVD robots in our system for very different purposes: we programmatically control the DVD robots to read or generate stacks of customized DVDs as the Postmanet-enabled web server handles incoming or outgoing data. This is a key innovation that allows us to significantly reduce the level of human intervention needed to keep the system running.

The recurring theme of the necessity of providing systems support that can make the Postmanet general, transparent, efficient, and scalable is best illustrated by our innovative use of the DVD robots: in terms of generality, the DVD robots should be resources that can be exploited by any application; in terms of transparency, the integration of the robots into the communication processes minimizes non-transparent human involvement; in terms of efficiency, the robots can process DVDs more quickly and more cheaply; and in terms of scalability, a robotically-aided Postmanet server should be able to handle many more Postmanet clients.

What we have seen in this subsection is that though appealing, making a browser-based application work on top of the Postmanet requires non-trivial systems support in terms of both software and hardware, whose level of sophistication is quite likely beyond what most browser-based application programmers can or should handle. We dub the layer of systems software that handles the responsibilities discussed in this subsection *PHTTP*, and we discuss it in greater detail in Chapter 4.

1.4 Routing

In the "routing problem," we ask how data travels from the sender to the receiver in the Postmanet. As we shall see, the answer needs to be a lot more sophisticated than "leave it to the postal system."

In our first deployment of the Digital StudyHall application, content produced by the urban headquarters school, the Study Hall school, is sent to several surrounding village and slum schools. The arrangement forms a simple hub-and-spokes model, where the resource-rich urban school is the hub, and the resource-poor village and slum schools are the spokes. Any communication in the system is "routed" through the database at the hub, so, for example, when Village A needs to send content to Village B, Village A sends the information to the headquarters school, which, in turn, forwards the information to Village B.

This simple arrangement has some important advantages. On any particular day, regardless how many other sites Village A desires to communicate with, it only needs to send the hub site a single outgoing movable storage device, which contains all the data destined for the multiple intended receivers. And similarly, regardless how many sites Village B receives data from on any given day, it receives from the hub site a single incoming storage device, which contains all the data originating from the multiple senders. The hub site is responsible for de-multiplexing data from incoming storage devices and re-multiplexing data onto outgoing devices. (In this way, the space on any storage device is also well utilized, and if the data that needs to be communicated exceeds the capacity of a single device, we may simply use more devices than one.) Minimizing the number of storage devices that must be handled by each site in this manner provides significant convenience to all those involved.

The question is how we intend to scale up the system to include many more villages, from a "networking" point of view, where the villages may desire to communicate with each other. A single hub-based solution has obvious disadvantages. The single hub can easily become a bottleneck. And when a pair of communicating villages are close to each other, the mandatory intermediate hop through the hub, which can be far away, introduces unnecessary extra delay. If we consider the DVDs transported by the postal system as the rough equivalent of the wires in a conventional network, the simple hub-and-spoke model is analogous to a star-shaped network, with its known limitations.

This is an important question that is more generally relevant to the underlying Postmanet system than the special case of the Digital StudyHall application built on top of it. To circumvent the disadvantage of a single bottleneck in the star network, an alternative model is to require individual communicating parties in the Postmanet to send storage media directly to each other, so, for example, Village A prepares an outgoing DVD for Village B and a separate one for Village C. In this direct "peer-to-peer" approach, we are essentially leaving the routing problem to the postal system. In terms of the DVD versus wire analogy, this peer-to-peer approach is roughly equivalent to a "complete graph," which provides a one-hop wire between *any* pair of nodes in the system. Such a complete graph, while not being limited by a single bottleneck, has a well-known scalability problem too: an N-node system requires N^2 wires, or, in the case if this approach is used by the Postmanet, as many as N^2 DVDs that are exchanged on any given day. Each site is forced to handle a large number of storage media, and each of which may contain only a small amount of data.

To address the deficiencies of both the “star network” and the “complete graph” in our search for a good Postmanet routing approach, we again consider an analogy in conventional networking. In a conventional network, a message sent by an end host travels through a number of intermediate “routers” inside the “network” before it is finally delivered to the intended receiver. By employing multiple routers, this approach does not suffer from the problem of a single bottleneck in the star network. And by allowing the wires to be shared by multiple pairs of end nodes, this approach does not suffer from the problem of N^2 wires in a complete graph.

The Postmanet equivalent of this intermediate router-based approach is the use of what we call “data distribution centers.” An end host of the Postmanet only directly interacts with the closest data distribution center as the end host sends/receives movable storage media to/from the data distribution center. As is the case in the star network, each end host sends at most one storage device and receives at most one device per postman visit. Each data distribution center performs “gather-scatter” operations as it “gathers” data from the incoming storage devices sent by multiple senders, and recopies (or “scatters”) data onto multiple outgoing storage devices, destined to next-hop data distribution centers (or end hosts). If we carefully choose the number and location of the data distribution centers, a minimum number of data distribution centers should share the load evenly, while minimizing the total delay experienced by data as it is forwarded on movable storage devices from one intermediate data distribution center to the next.

Such data distribution centers in the Postmanet can take on several forms. Ideally, they should be embedded inside the existing postal system for maximum efficiency, as mechanical routing (via physical transportation vehicles) and digital routing (via gather-scatter data copies) are carried out by the same organization, so that these two types of routing can be optimized in an integrated fashion with full information of both.

Alternatively, these data distribution centers can be parallel structures set up independent of the existing postal system; and they may (or may not) choose to leverage the existing postal system simply as a transport vehicle among themselves. This second alternative of setting up a network of data distribution centers as a Postmanet data forwarding mechanism independent of the existing postal system has a particularly appealing practical synergy with respect to the Digital StudyHall application built on top of the underlying Postmanet system: these data distribution centers in the underlying Postmanet become co-located with content production hubs in the Digital StudyHall application, each of which is responsible for working closely with the poor spoke schools in its vicinity, producing content meaningful for the local context, monitoring the progress of the spoke schools, in addition to acting as intermediate data forwarding agents for the rest of the network.

A third alternative is to require Postmanet end host nodes to shoulder some of the responsibilities otherwise taken by the data distribution centers. For example, a disc sent from Village A to Village B may contain data destined for both Village B and Village C, and therefore, Village B is required to pass the disc onto Village C. This is analogous to the use of end hosts in a conventional network as “overlay routers.” This approach has some important advantages. First, it may reduce the need for many strategically well-placed data distribution centers. Second, it reduces the number of discs any one site may need to handle in absence of well-placed data distribution centers: in the simple example above, Village A sends out only one outgoing disc despite the fact it desires to communicate with two destination villages (B and C). A good “overlay routing” algorithm in the context of forwarding discs in the Postmanet, however, is a non-trivial problem. If we do not exercise sufficient care, a node may be forced to handle too many discs (destined either to itself or others), or data may be forwarded across too many postal hops (thus lengthening overall delay), just to name a few of the possible problems. Existing ad hoc routing strategies [43, 48, 27, 47, 55] are not adequate for our purposes. Unlike ad hoc or sensor networks, the Postmanet routing challenge is not route discovery but route selection. More specifically, the task we face is not to *discover* a route through a small number of neighbor nodes in an ad hoc network; instead, our task is to *select* a route in a complete graph where *all* routes are possible, subject to constraints such as end-to-end latencies and the numbers of discs handled per node.

From the preceding, we can conclude that an ideal Postmanet routing mechanism should possess the following characteristics: (1) it can accommodate a large number of simultaneous Postmanet communicators without requiring a site to handle many mobile storage devices per postman visit; (2) it has end-to-end message propagation latencies that are close to those provided by the postal system; (3) it does not require an expensive infrastructure other than the existing postal system; (4) it does not burden Postmanet nodes in an unbalanced manner with data copying tasks that are beyond their own communication needs; and (5) it is robust when faced with misbehaving Postmanet end users. Some of these goals are unique to the Postmanet. These goals often conflict with each other and we need to strike a proper balance among them. Good routing solutions form an important part of the system support that is essential in order for the Postmanet to achieve its generality, transparency, efficiency, and scalability goals. We discuss our solutions to the routing problem in chapter 5.

1.5 A Message Passing API and Its Implementation

As discussed in the previous subsections, our example Postmanet application, the Digital StudyHall system, is built on top of the PHTTP layer, which provides systems support for browser-based Postmanet applications. During about seven months of production use by testbed schools in India, PHTTP and the Digital StudyHall have allowed us to gain experience and insight into some unique challenges presented by the Postmanet that should be addressed.

In a conventional "always-on" network, data transmission may be on-demand and continuous. In an asynchronous communication mechanism like the Postmanet, however, upon the receipt of an incoming storage device, the data receiver may face the sudden arrival of a large amount of data. For example, each MPEG4 model lesson video used in the Digital StudyHall system is typically 200-500 MB in size. An incoming DVD received by a participating village school may contain around a dozen such videos; and the hub school may receive DVDs filled with monitoring videos captured at dozens of spoke schools. Naturally, the operators and teachers frequently desire to quickly gain access to a coherent view of the database with the new incoming data integrated. A naive solution that forces the operator to wait while the system is busy copying many gigabytes of data from incoming storage devices into the local database can be frustrating. The challenge is to devise a system that permits operations on a coherent database view whose data may be spread across multiple storage devices, some of which are the movable storage media being copied.

Some of the village schools participating in the Digital StudyHall system are closer to urban centers and have cellphone signal coverage, which can be exploited as the LLLB channel that complements the HLHB postal channel. (In India, receipt of incoming cellphone calls is free. It is possible to establish a Postmanet LLLB link by exploiting this payment policy, thus minimizing the cost.) While the bulk data of the recorded lesson videos distributed by the Digital StudyHall must be carried by the postal link, smaller data elements, which include both control messages and data items, can be more effectively carried by the cellular link. Examples include receipt acknowledgements or negative acknowledgements, metadata, “abstracts,” or “excerpts” detailing what content is available at each site, requests for data, search requests and results, systems logs and other monitoring information, and trouble-shooting queries and responses. Our experience indicates that we may significantly improve the user experience if these small messages are carried by a complementing alternative LLLB link instead of relying on the HLHB postal link. For places that lack cellular coverage, we are exploring the use of ham radios. The communication software on top, though, should be neutral to the choice of the physical connectivity mechanism underlying the LLLB link. Making the Postmanet system, and more generally, the distributed database on top of it, work on multiple underlying connectivity mechanisms simultaneously is another challenge.

In an asynchronous communication system, there can be significant delay between the time when a message is sent and the time when the message is consumed and acted upon by the receiver. During this delay, the message can be at any one of many possible locations, including in-transit locations somewhere “inside” the network, while circumstances could have changed to make continuing the delivery of the message no longer necessary. For example, the sender might have changed his or her mind and would like an “undo” or “cancel” operation; or alternatively, later operations (such as repeated rewrites of the same data) could have made earlier operations (such as the earlier writes) unnecessary as “coalescing” or “consolidation” of multiple messages should occur during the transit of multiple messages. Simple examples in the Digital StudyHall system include cases where newly updated versions of class videos make earlier in-transit incarnations obsolete. Such undo or coalesce operations are less critical in a conventional network where the typical in-transit time is small. In a “network” like the Postmanet, the opposite is true; and depending on the application, a large fraction of the traffic could fall into this category. These undo or coalesce operations are not only useful for performance reasons; but also desirable from the point of view of user functionality.

In the Postmanet, storage devices play the role of wires in a traditional network. Not much “data structure” beyond a sequence of bytes is employed on a wire. In contrast, there exists a large body of research examining how careful design of on-disk data structures can significantly improve the performance of traditional file systems. What data structures we should employ for the movable storage devices used in the Postmanet system and what access interfaces we should support are natural questions. The functionalities that the Postmanet discs need to support straddle those required of traditional wires and traditional storage devices: they need to support both efficient sequential copying (based on temporal ordering) and efficient non-sequential access (as operator actions at end hosts result in selective access to the movable storage devices before their data is fully integrated into the end host store).

The challenges described above: the massive data size that is involved, the long delays associated with both in-transit times and copying latencies, the need for undo or coalesce operations, the need for employing intelligent data structures on the movable storage devices, are unique to the Postmanet system underlying an application like the Digital StudyHall. Even just from a performance point of view, as an example, the need of an intelligent system-level data copier that can exploit its knowledge of storage device layout, that can support and coordinate needs from multiple applications, is a non-trivial challenge. Addressing these challenges, a task that is more complex and more general than that should be left to individual application programmers, is yet another important part of the systems support that we need in order to achieve the generality, transparency, efficiency, and scalability goals of the Postmanet system.

In this part of the study, we have chosen to experiment with a different software layer than the PHTTP layer, for several reasons. One is that PHTTP is designed to support browser-based applications, and for this part of the study, we desire to provide a lower-level interface that provides us a greater degree of flexibility. As a result, in this part of the study, we gain the freedom of being able to design a new API as a first step of addressing the issues discussed in this subsection. We discuss this API, dubbed *pnet0*, and its implementation in Chapter 6.

1.6 Thesis Contributions

We summarize the main contributions of this thesis: (1) inventing the approach of turning the postal system into a generic and transparent digital network, the Postmanet; (2) demonstrating a compelling and sophisticated real-world example application, the Digital StudyHall; (3) exposing the systems support needs that are critical to meeting the generality, transparency, efficiency, and scalability goals of the Postmanet, including those related to end point APIs and infrastructural support (such as routing); (4) devising and evaluating a number of practical solutions to these systems support problems, in the forms of the PHTTP and pnet0 APIs, and the various routing algorithms that are based on leveraging data distribution centers and peer-to-peer forwarding, and must account for unconventional Postmanet-specific routing metrics. We do not claim that the practical solutions that we have devised are optimal. Instead, we see them as first-cut enabling building blocks that allow us to demonstrate the feasibility and power of the Postmanet approach, especially in terms of its potential in bringing about positive changes in impoverished regions of the world.

Chapter 2

Postmanet: Usage and Applications

An important goal of ours is to make the Postmanet as transparent as a conventional network channel for a user. One way of better understanding this transparency is to visualize a box that is similar to a small conventional home network router: it allows several home computers to share a wide area Internet connection. A Postmanet router (or a *P-router*, illustrated in Figure 2.1) is just such a box with one or more slots for inserting mobile storage media such as DVDs.

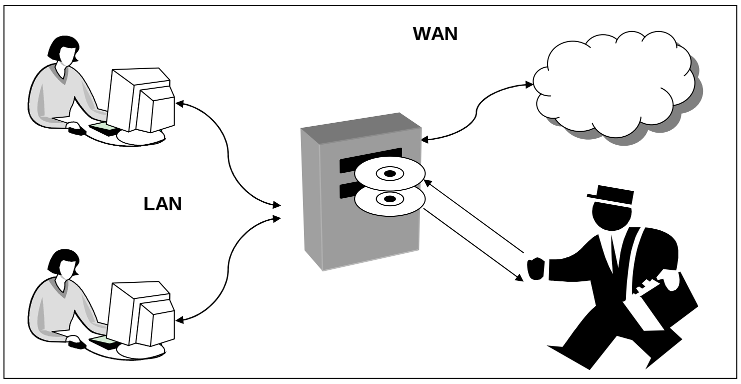


Figure 2.1: A Postmanet router.

During the day, a user of the Postmanet router simply uses his applications in a way that is entirely oblivious of the presence of the Postmanet. Instead of always forcing outgoing data through a weak wide-area network, however, the P-router writes some of the outgoing data to a mobile storage media. The types of storage media used may include write-once or read-write DVDs, flash memory cards, or hard disks. We shall generally refer to these storage devices as *P-disks*. At the end of the day, the Postmanet router box automatically spits out an outgoing P-disk filled with some data. An outgoing P-disk, after being ejected from a P-router, is picked up by a postman for delivery via the postal system. The postman may also drop off an incoming P-disk. The user does not have to manually inspect or process the content of the P-disk in any way: he just inserts the P-disk into a slot in the Postmanet router box. From that point on, the user continues to use his applications in an oblivious way, and the P-disk's data appears on a user computer as if it had arrived from a conventional WAN.

The details of this imaginary Postmanet router box can vary. The box may not necessarily be a dedicated physical device: the user home computer may shoulder the task or it could be a box shared by an entire neighborhood placed next to the regular mailboxes of the community. The number of P-disks picked up and dropped off by the postman per visit may vary. The Postmanet router box may or may not be complemented by a conventional wide-area network connection. The box may or may not be shared by multiple users. While all these details may vary, a constant is the transparency feature: the fact that the user's direct manual interaction with the box is limited to the insertion and removal of one or more P-disks per postman visit.

We emphasize here that when we say that the P-router's function in the Postmanet is equivalent to that of a conventional router in a conventional network, we do not mean that it provides the same interface to applications and user experience as a conventional router.

We do not mean that existing applications can run unmodified over the Postmanet. The P-router provides an API which the application developers use to make the user experience work. The transparency, in other words, is from the user's point of view and not from the point of view of the *existing* applications. The Postmanet and the applications work in conjunction to provide users with an experience similar to that of the equivalent existing applications that only use the conventional network.

2.1 Postmanet Router Challenges

The P-router must perform several "system level" tasks to provide a generic and transparent interface and efficient performance. We now examine some of the challenges involved and the opportunities that can be exploited to perform these tasks.

2.1.1 Simultaneous Exploitation of the Internet

The Postmanet can be very valuable in absence of any traditional connectivity. With the aid of an LLLB connection such as a phone modem, however, the Postmanet becomes even more powerful and interesting. However, the applications should not have to worry about which of the Postmanet channels (LLLB or HLHB) to use to meet the users' needs; when data arrives at a Postmanet receiver via the postal system, for example, the receiver should send an acknowledgement back to the sender over the Internet. This may further cause the sender to discard a local data copy that may have been saved for potential retransmission. More generally, the sender system may choose between the LLLB Internet and the HLHB Postal network based on factors such as the amount of data to be sent and the desired arrival time. Indeed, the system may choose to use the LLLB and HLHB channels in parallel. Portions of a large data object may start to incrementally arrive at the receiver over the Internet, while the complete object arrives later over the postal network. At the application level, multiple versions of a data item may be prepared: for example, a low resolution version is shipped over the LLLB Internet, while a high resolution version is shipped simultaneously via the HLHB Postal network. Multiple versions of the data may “race” against each other as they progress in the two different “networks,” so we can trade off metrics such as quality, latency, and availability. If the Internet is “max’ed out,” the system needs to carefully prioritize what is sent through the Internet connection. One way of looking at this problem is to view the Internet connection as a “cache” of the Postmanet connection: the former is a faster (latency-wise), smaller (bandwidth and capacity-wise), and sometimes more expensive alternative that provides comparable functionalities. The question is how to use this scarce resource in an appropriate way.

2.1.2 Liberal Exploitation of Excess Capacity

In addition to possibly sending redundant data simultaneously over the Internet and the Postmanet, we may also proactively replicate data in the Postmanet. For example, as multiple P-disks are sent between a sender-receiver pair on successive days, we may liberally replicate outgoing data of earlier days on outgoing devices sent on later days. In cases where a single P-disk is delayed or lost due to accidents in the postal system, the replicated data on subsequently arriving devices is just a day away, so we can avoid unnecessary long end-to-end retransmission delays. The opportunity to liberally “waste” plentiful resources (storage capacity) to optimize for more difficult metrics (lower latency or better reliability) needs to be carefully managed however.

One possible factor that may constrain the liberal copying of data onto a P-disk at the sender is available time. If the P-disks being used can be written to incrementally, we may not need to wait till shortly before the arrival of the postman to begin writing to the device in a long burst – continuous background copying could have occurred throughout the day. At the application level, if a later sending event should supersede earlier ones (because, for example, only the freshest version of an updated object needs to be sent), the system would take care of excluding from the P-disk obsolete data that is sent earlier. Also, the "wasting" of storage capacity has to be done intelligently. Once a data item has reached its destination, the still in-transit copies of the data should be discarded and the reclaimed space reused.

2.1.3 Handling Datagram Limitations

The limitations of postal system datagram delivery are exacerbated by our aggressive exploitation of the Internet and the excess capacity. At the systems level, for example, due to proactive replication or premature retransmission by the sender, either across multiple P-disks, or across the Internet and the postal system, the receiver may need to discard the duplicates. Multiple P-disks may have been delivered by the postal system out-of-order; and data delivered by the postal system and by the Internet may arrive out-of-order. Similar issues may occur at the application level also. For example, even in absence of duplicates or out-of-order delivery, the receiver application may discover that some of the newly arriving data is no longer needed due to application-specific reasons. In all these cases, the system must exercise care not to unnecessarily copy or use obsolete data. This is especially relevant due to our general approach of liberally "wasting" storage capacity in an effort to optimize for other metrics—this “wastage” needs to be checked by cleverly combining application-specific intelligence with Postmanet’s transport-level algorithms.

2.1.4 Security and Other Issues

Security is another issue that the Postmanet must deal with. The receiver may desire to ensure that (1) the incoming P-disk is from a sender whom it is willing to receive data from; (2) the sender identity is not forged; (3) the data has not been tampered with; and (4) data that should have been private is not disclosed to eavesdroppers. The sender may need to compute fingerprints and/or encrypt data on outgoing P-disks. A P-router should be able to authenticate a shoot-down request as being legitimate.

In addition to resolving transport-level issues, we also need to provide easy-to-use APIs. Programming models similar to existing asynchronous communication models [64] and the programming languages built on top of them [10] may be desirable. Under these models, handler codes associated with messages are asynchronously executed upon arrival of the messages to incorporate the newly arriving data into ongoing computations. Applications can be granted direct access to the data contained on the P-disk to make data copying out of the mobile storage device potentially unnecessary. For read-only P-disk media, copy-on-write techniques may be necessary.

Many of the issues described above, such as retransmission, handling out-of-order delivery, suppressing duplicates, and minimizing data copies, bear a resemblance to those that one must deal with in traditional communication networks. In the context of the Postmanet, however, not only are these problems further complicated by our aggressive exploitation of the Internet and the excess capacity, it is also the case that the boundary between storage and networks is blurred. The latency and the amount of data involved in a “packet” (i.e., a P-disk) can be many orders of magnitude greater than those of a traditional network packet. This makes the problem as much a distributed storage problem as a networking problem.

2.2 Applications That Can Benefit From Postmanet

We now consider possible applications that can benefit from utilizing the Postmanet network. The following example applications share at least two common themes: (1) their bandwidth demands can far exceed those that can be met by a traditional wide-area network; and (2) these applications can benefit from the *simultaneous* exploitation of the HLHB Postmanet and the LLLB Internet.

• *Distance learning.* In addition to multimedia teaching material that is being disseminated by teacher sites to students over the HLHB Postmanet, the students may submit content such as digitized homework for grading over the Postmanet, and the resulting teacher feedback may be sent back over the Postmanet again. Smaller network messages such as teachers’ synchronous commands controlling the real-time playing of teaching material at students’ sites may be transmitted over the LLLB Internet. We discuss one such distance learning application, called “The Digital StudyHall,” in detail in chapter 3.

• *Email with large attachments.* For example, one may be able to send large home movie files via email. This application may take advantage of the LLLB Internet by sending the small message body over it, while the large attachments travel over the HLHB Postmanet. (Certain extra UI features are needed to deal with the decoupled arrival.)

• *Web pointing to or embedded with large data objects.* These large data objects may include audio, video, programs, and models. To avoid the need of client-side browser modification, one can employ a Postmanet-aware client-side proxy. Small data items, such as top-level html pages, can be retrieved over the LLLB Internet (to ensure their freshness). Large data objects that are pulled or pushed over the HLHB Postmanet can be placed in a client-side cache. The client-side proxy may poll over the LLLB Internet to check the freshness of the cached data. There are two possibilities for the server side: the content publisher is either Postmanet-aware or not. A Postmanet-aware content publisher program may respond to client "subscription" requests by sending them large data items over the Postmanet. For a content publisher site that is not Postmanet-aware, a possible way for its large data items to reach poorly-connected clients is via a well-connected third party that *is* Postmanet-aware: this third party would retrieve large data items from the original content publisher over a conventional network and repackage them to send to poorly-connected subscribing clients over the Postmanet.

• *Remote file system mirroring for sharing and/or backup.* Large amounts of newly written file data can be transmitted to a remote mirror site over the HLHB Postmanet. Users at this remote mirror site who desire to read up-to-date versions of the files may use the LLLB Internet to check freshness of the mirror. Also, the P-disks sent over the HLHB Postmanet, if not the rewritable kind that are immediately erased and reused, can be safely stored in a vault and act as an additioinal archival data copy, providing additional data reliability or reducing the number of hard disks that are needed to guarantee a certain degree of reliability.

• *Peer-to-peer file sharing.* Large media files are excellent candidates for transmission over the HLHB Postmanet. The small messages generated by foreground file searches or background announcements of sites' contents can still use the LLLB Internet. Note that the use of the Postmanet is orthogonal to the choice of the overall file sharing system architecture, which can be based on either centralized metadata servers or entirely decentralized alternatives. Copyright protection concerns can be addressed by incorporating

Digital Rights Management (DRM) techniques [2] and the Postmanet should be neutral to such concerns, just as a traditional network is.

• *Video "almost on-demand."* A shortcoming of the existing online DVD movie rental businesses is the multi-day latency elapsed between the time a request is submitted over the Internet and the time the desired movie arrives via the postal system. In an alternative model, subject to customer permission, the rental company could proactively push encrypted movies to participating customers without necessarily having received explicit requests. These may include recommended movies based on customers' rental history, popular movies, and new releases. At the current rate of storage density and price improvement, it would be very reasonable to assume multi-terabyte hard disks filled with hundreds or even thousands of movies being employed by the HLHB Postmanet. Large encrypted libraries of movies can accumulate on participating customers' local storage devices. To view a movie, a customer would purchase a decryption key on-demand from the rental company over the LLLB Internet and gain access to a locally stored and encrypted selection instantaneously. Again, emerging DRM technologies such as Microsoft's Palladium [2] should be able to prevent unauthorized dissemination of decrypted content or other usage that is outside a contract. For example, such a DRM contract may restrict the number of times that a movie can be played for a certain payment.

• *Publish/subscribe systems for other types of content.* The above video "almost on-demand" application can be generalized to disseminate many other types of content in a generic publish/subscribe system. The types of content may include music, TV and radio programs, newspapers, magazines and store catalogs (with richer presentation), software releases and updates, and public lectures given at universities. The possibilities enabled by an inexpensive communication channel with practically infinite bandwidth can be vast. While it is possible to develop and maintain individual solutions for different types of content, the presence of a *generic* Postmanet infrastructure that is available to all applications makes the approach more attractive.

In the above discussion of example applications, we have seen how we can exploit the simultaneous use of the HLHB Postmanet and the LLLB Internet. The availability of an LLLB Internet, however, is not absolutely necessary for the functioning of the HLHB Postmanet. For example, a user may receive a large digital catalog of Amazon.com via the Postmanet, browse the catalog and place purchase orders “off-line,” and send back the orders via the Postmanet (instead of via an Internet connection, had it been available). This arrangement is especially useful for places such as isolated remote regions in developing countries where even dialup connections are not always available. (Yes, even the economically disadvantaged have purchasing power, and it is often useful being able to transact with sources outside of their confined neighborhoods that may offer limited choices and/or artificially inflated prices[51].)

Although we have called the postal link a “high latency” channel, we note, once again, that by exploiting the plentiful storage capacity and bandwidth of the Postmanet, it can be possible to mask its high latency. The video “almost on-demand” example is particularly illustrative: by liberally disseminating content that may never be actually used by users who receive it, a publisher who uses the HLHB channel can, perhaps ironically, create the illusion of instantaneous on-demand access for content that *is* used. This theme of deliberately “wasting” plentiful resources to optimize for scarce resources will be revisited in other aspects of the functioning of the Postmanet.

Chapter 3

The Digital StudyHall

The Digital StudyHall is a substantial example application of the Postmanet system. It is a distance learning system that aims to "connect" resource-starved "spoke" schools in rural India to resource-rich "hub" schools in urban environments, so that the poor rural schools can benefit from the better human, content, and methodology resources available in the better equipped counterparts in cities (Figure 3.1). A database located in a good hub school at the center collects digital content such as recordings of live classes taught by the best teachers, coursewares and course materials. These materials are transmitted to the surrounding rural schools, where the students and teachers utilize them to improve their learning. The rural schools may submit locally created materials (such as student projects and recordings of their own lessons recorded for monitoring and feedback purposes) to the hub database. And the hub school may send out further feedback. As all these schools are benefiting from the same high-quality digital "diet," our hope is that this continuous digital dialog would narrow the urban/rural, private/public, rich/poor school gaps. The pervasive, cheap, high-bandwidth connectivity provided by the Postmanet is unmatched by any other connectivity option available today or in the foreseeable future in rural India.

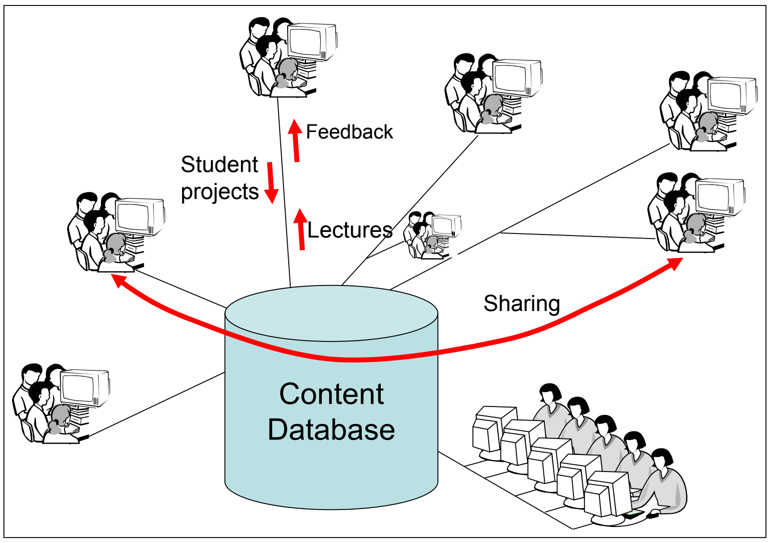


Figure 3.1: The hub-and-spoke model. An digital database in an urban school is "networked" with a number of village counterparts to improve rural education.

3.1 A Network of Hubs and Spokes

Over time, we would like to scale up the system to bring in more villages, more schools, and more children. Our approach to our scale-up goals is to build a network of hubs and spokes (Figure 3.2). A hub is typically located in a center of teaching excellence in an urban area; and its purpose is to "radiate" locally relevant content and methodology into the slum and village schools in its neighborhood. The multiple hubs are themselves networked with each other. In addition to the postal links, unlike the village schools, the hub schools typically also have access to broadband connectivity, although the available broadband bandwidth is still much smaller than that is required if one were to attempt to directly transmit the multimedia content on the broadband connections. Out of this set of initially disjoint hub databases, we would like to join them together into a coherent distributed database, not unlike some of the existing peer-to-peer file sharing systems

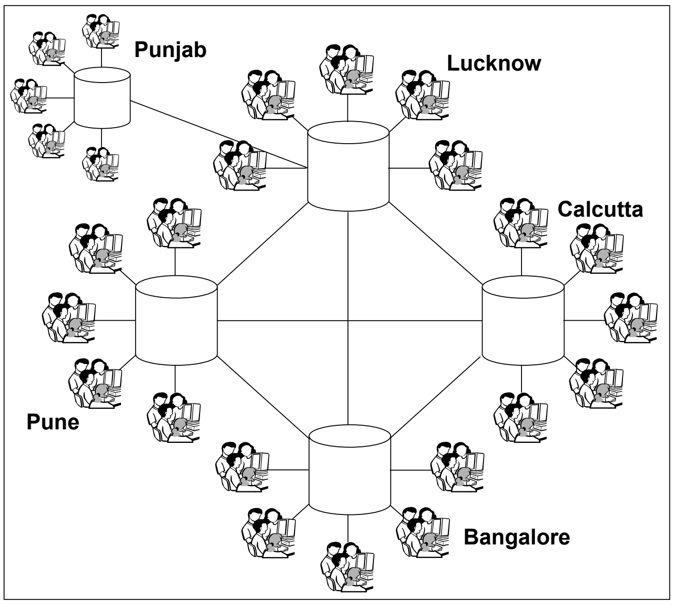


Figure 3.2: A network of hubs and spokes. Each of the networked centers of excellence in urban settings "radiate" content and methodology into their neighborhood village and slum schools.

such as Gnutella [19]. Of course, the added challenge in our case is that our system is built on top of two different types of underlying connectivity technologies.

This model of a network of hubs and spokes is appealing from the points of view of networking, systems, and non-technical considerations. First, at the lowest level, from the networking perspective, each of the hubs acts as a Postmanet data distribution center (P-center), a role that is analogous to the routers and switches in a conventional network. They constitute parts of a scalable routing infrastructure that allows an end node in a village or a slum school to effectively communicate with the rest of the Postmanet nodes. Each end node sends to and receives from only the hub that is closest to it. A hub de-multiplexes data from incoming P-disks and re-multiplexes them onto outgoing P-disks, destined for other hubs, which, in turn, forward data onto the spoke nodes attached directly to them. Just as the wires and the switches in a conventional interconnected network (or Internet) that shoulder traffic between many possible pairs of end hosts, the

P-disks and the P-centers become shared resources, so we minimize the number of P-disks that need to be handled by any site, within the constraints of end-to-end latency and P-center load requirements.

Second, from the point of view of a peer-to-peer file sharing system, the model of a network of hubs and spokes is appealing, as it provides a large degree of flexibility in terms of addressing questions such as where data is physically stored, how many copies are kept, from which site to get the data from if one does not have it locally. The resolution of such questions is vastly simplified due to the presence of the "second link," the LLLB broadband link among the hubs.

Third, from the point of view of education content production and sharing, the model of a network of hubs and spokes allows us to strike a proper balance between the need of locally relevant content and a powerful "network effect" as good content and practice can become widely shared. The need of ensuring local relevance is particularly important in India, as each region has its own unique local language, and almost all the government public schools conduct almost all their instructions using the local dialect. On the other hand, mastery of English is a particularly sought-after skill because it can open many more doors in terms of employment opportunities. In general, a good command of English is an empowering tool in India. But there is an acute shortage of village teachers who can speak any English at all. This is an example of a subject that can benefit from the wide spread of good content and pedagogy made possible by easy sharing across a network of many hubs.

Fourth, the model of a network of hubs and spokes is also appealing from the point of view of fruitful collaboration between the Digital StudyHall (DSH) project and the various non-government educational organizations (NGOs). We have had enthusiastic responses from virtually every educational organization that we have discussed with. The decentralized arrangement of a network of hubs and spokes empowers the existing NGOs to amplify their reach and voice. DSH does what it does best: providing the plumbing of an effective communication infrastructure; and the NGOs do what they do best: working with total freedom on educational content, pedagogy, and communicating with people; so DSH and the NGOs are helping out each other in a very synergistic relationship. The NGOs are enthusiastic in joining the DSH network and making it successful, not necessarily because they want to make DSH per se successful, but because it magnifies their ability of doing what they have always wanted to do: reaching more students than they could have otherwise, with both their own voices and the best practices they get from colleagues elsewhere.

Even at this early stage of the Digital StudyHall development, we are already actively working on establishing multiple hubs, beyond our first pilot hub in Lucknow. The impetus is that we would like to involve the parallel efforts in content production and pedagogy research by other NGOs as early as possible. By the end of this year (2006), we aim to establish at least five hubs (shown in Figure 3.2): Lucknow, Calcutta, Pune, Bangalore, and Punjab. Each of these hubs specializes in a different local language: Hindi in Lucknow, Bengali in Calcutta, Marathi in Pune, Kannada in Bangalore, and Punjabi in Punjab. We collaborate with existing NGOs in each of these cases, and they are at various different stages of development at the time of this writing (May of 2006), with the Calcutta hub having made the most progress at this time outside our pilot hub in Lucknow.

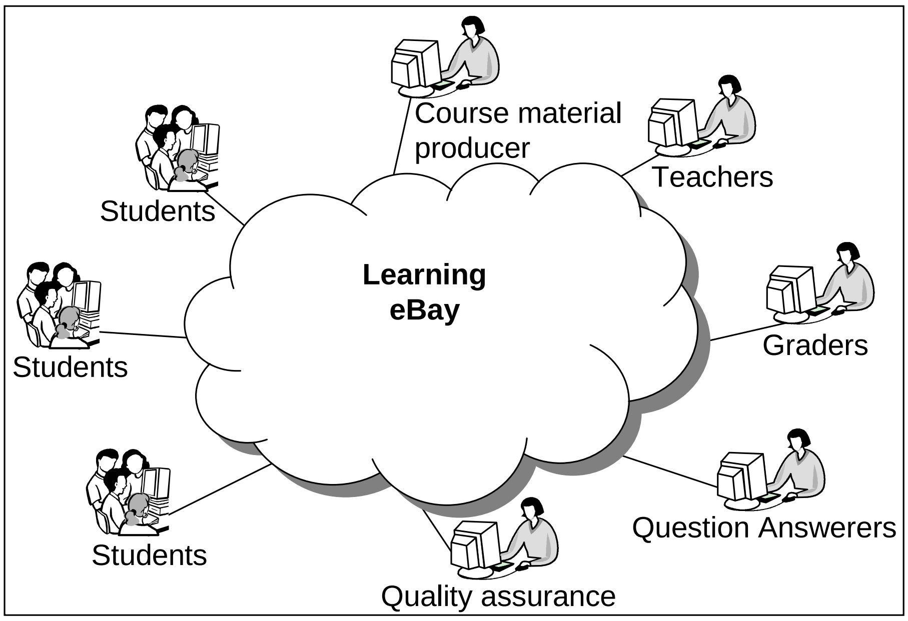


Figure 3.3: The "Learning eBay." The network of hubs presents a coherent logical view of a single repository that allows people to collaborate across the network.

3.2 The Learning eBay

While the network of hubs and spokes constitutes the *physical* structure of a distributed database, *logically*, in the longer run, we would like to develop it into a coherent repository as well as a meeting place of learners and teaching staff across time and space (Figure 3.3). Volunteers (who may choose to work for free) and professionals (who may "teach" on the system for a fee) "plug" themselves into the system to play various roles. Some may develop teaching materials and make them available on the site. Some may use other people's materials and conduct teaching sessions. Some may grade homework assignments that students have submitted to the site; the graded results are available for students to download. Some may conduct virtual "office hours." Students and staff with limited experience or resources may tap into this online site to enrich or start their local schools. It is also a way for teachers to get trained.

We dub this site a “Learning eBay[1],” in the sense that this is a virtual meeting place and a “market place” that aggregates and matches “supplies” and “demands,” supplies and demands for educational resources and services. Such a site would allow teaching professionals and volunteers to make flexible time and location commitments: for example, a volunteer can decide to spend perhaps only four hours a week grading some homework, potentially from his home overseas. This arrangement may allow us to address the difficult issue of attracting and retaining well trained and qualified teaching staff in remote regions. It may also allow us to build an education system that is analogous to an “open source” model, in which legions of content developers all over the world constantly pool their contributions towards a single coherent repository that is accessible to all. It may also allow parts of the operations of a traditional school to be operated based on an “outsourcing” model: for example, homework grading from villages can be standardized and “outsourced” to a centralized remote location (at Study Hall in Lucknow, for example). Such an outsourcing model may address staff shortage in remote areas, ensure uniform and high standards of the outsourced operations, increase specialization and efficiency.

3.3 The Content Database Interfaces

Having painted the broad strokes of how the “big picture” of the Digital StudyHall network functions, we now turn to the specifics of the user interfaces faced by operators at the hubs and at the spoke schools. (We will refer to these as the “hub interface” and the “village interface.”) In this section, we discuss only some interface issues and the user

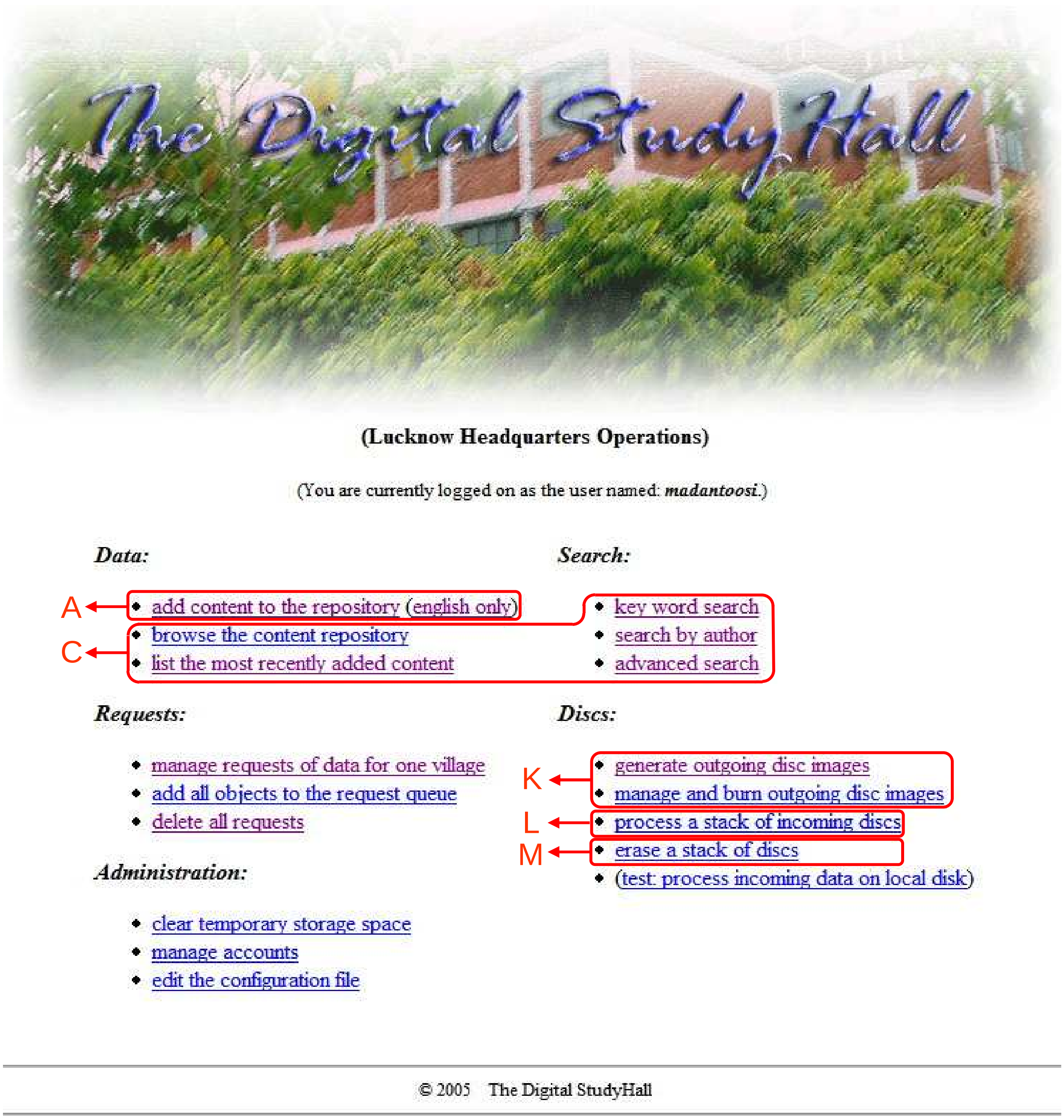
[1]While eBay is synonymous with online auctions for the purpose of making money, we use the word to describe a meeting point for demand and supply for educational content and services. We expect most of the content on the site to be provided for free.

experiences, and we leave the implementation issues for later. As far as the interfaces are concerned, we hope to convey two themes: (1) the interfaces are simple to use; and (2) they attempt to present a view of the database that is as uniform and coherent as possible, regardless from where and with what connectivity technology a user attempts to access the database.

3.3.1 Operation Overview

It is useful to consider the following three dimensions. (1) Who is the operator? We distinguish between a "hub operator" and a "village operator." A hub operator has more complex responsibilities than a village operator. (2) What connectivity technology is the operator using? An operator could be utilizing `localhost` (i.e., accessing a standalone machine at the "console"), the Internet, or the Postmanet. (3) Which site is the operator attempting to communicate to? The answer could be a hub site or a village site. (In the ideal case in the long run, an operator perhaps should not even be aware of the concept of a "site," and this question may become meaningless in the future. In the current incarnation of our system, however, the operator does need to be aware of which site he or she is explicitly attempting to influence.)

All the operator actions are carried out through a web browser interface. A hub operator is greeted with the entrance page shown in Figure 3.4; while a village operator sees the front page shown in Figure 3.5. The entrance points to the data access interfaces under the "Data" and "Search" categories are substantially similar in the two interfaces. In terms of the three dimensions of the types of operations, the following possible combinations are the most frequently used. (1) A hub operator sits at a hub machine console, and changes the content of the hub database. (2) A hub operator sits at a remote site, and changes the



(Lucknow Headquarters Operations)

(You are currently logged on as the user named: ***madantoosi***.)

Data:

- add content to the repository (english only)
- browse the content repository
- list the most recently added content

Search:

- key word search
- search by author
- advanced search

Requests:

- manage requests of data for one village
- add all objects to the request queue
- delete all requests

Discs:

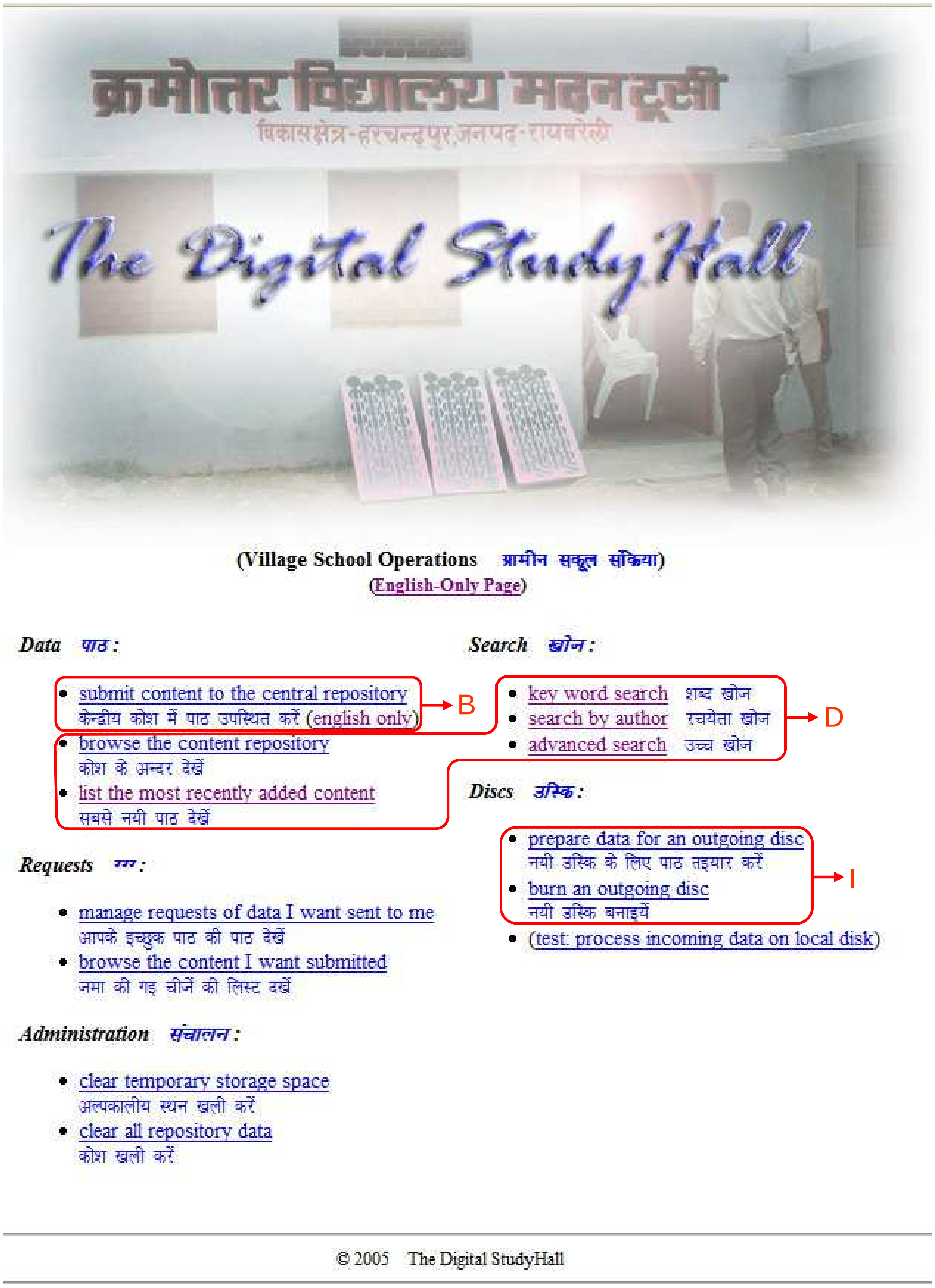
- generate outgoing disc images
- manage and burn outgoing disc images
- process a stack of incoming discs
- erase a stack of discs
- (test: process incoming data on local disk)

Administration:

- clear temporary storage space
- manage accounts
- edit the configuration file

© 2005 The Digital StudyHall

Figure 3.4: The hub operator entrance page.


क्रमोत्तर विद्यालय मदनटूसी
विकासक्षेत्र-हरचन्दपुर,जनपद-रायबरेली
The Digital StudyHall
B
D
I

(Village School Operations ग्रामीन सकूल संक्रिया)
(English-Only Page)

Data पाठ:

- submit content to the central repository
  केन्द्रीय कोश में पाठ उपस्थित करें (english only)
- browse the content repository
  कोश के अन्दर देखें
- list the most recently added content
  सबसे नयी पाठ देखें

Requests ग्ग:

- manage requests of data I want sent to me
  आपके इच्छुक पाठ की पाठ देखें
- browse the content I want submitted
  जमा की गइ चीजें की लिस्ट दखें

Administration संचालन:

- clear temporary storage space
  अल्पकालीय स्थन खली करें
- clear all repository data
  कोश खली करें

Search खोज:

- key word search शब्द खोज
- search by author रचयेता खोज
- advanced search उच्च खोज

Discs डस्कि:

- prepare data for an outgoing disc
  नयी डस्कि के लिए पाठ तइयार करें
- burn an outgoing disc
  नयी डस्कि बनाइयें
- (test: process incoming data on local disk)

© 2005 The Digital StudyHall

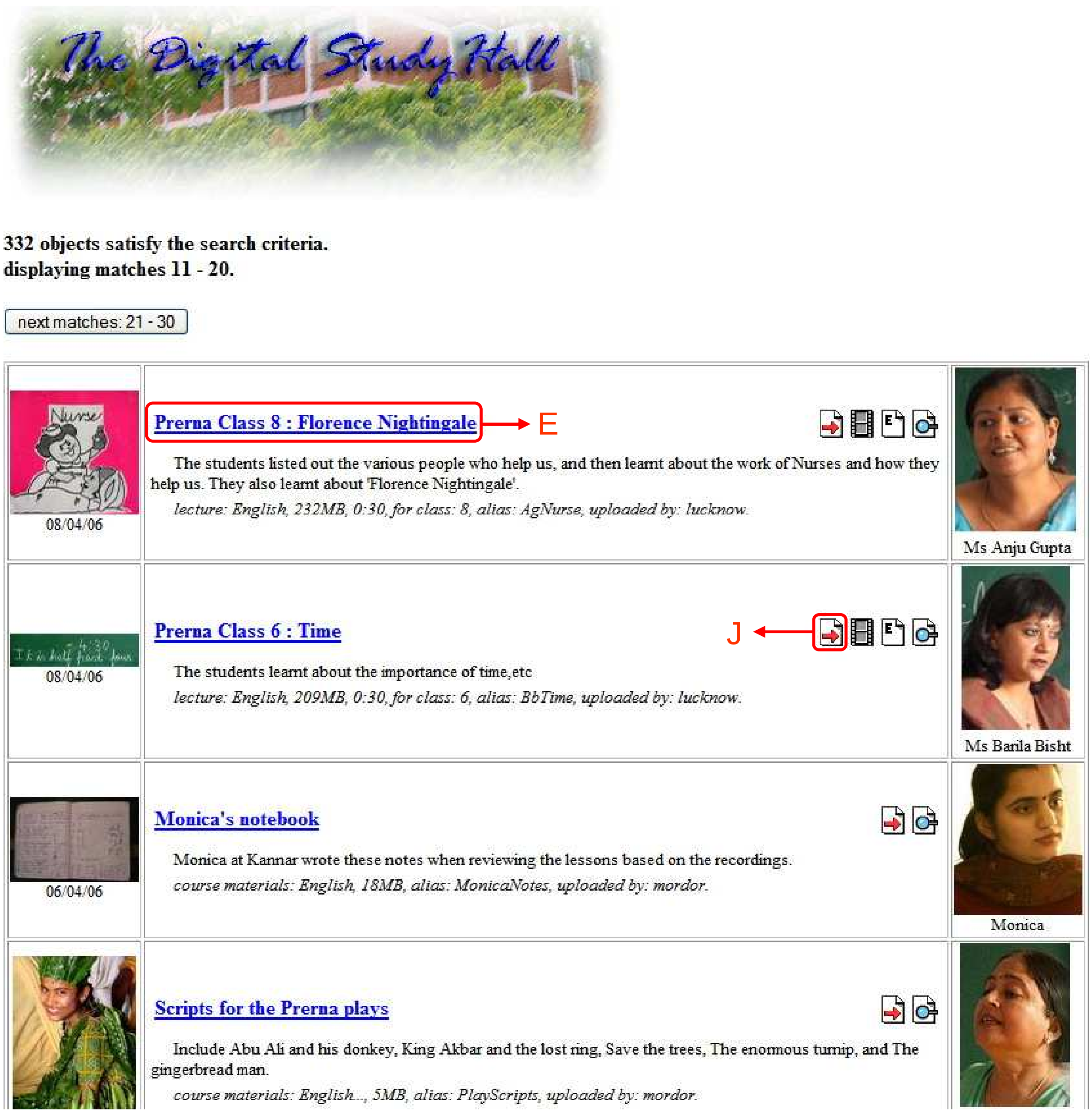
Figure 3.5: The village operator entrance page.

content of the hub database via an Internet connection. (3) A hub operator sits at a hub machine console, and initiates communication with a village site via the Postmanet. (4) A hub operator sits at a hub machine console, and initiates communication with another hub via the Postmanet. (5) A hub operator sits at a remote site, connects to the hub site via the Internet, and initiates communication with a village site via the Postmanet. (6) A hub operator sits at a remote site, connects to the hub site via the Internet, and initiates communication with another hub site via the Postmanet. (7) A village operator sits at a village machine console, and changes the content of the village database. (8) A village operator sits at a village machine console, and initiates communication with a hub via the Postmanet. In the rest of this section, we use the example notation of "Fig 3.4-A" to denote an item that is labeled with the letter A in Figure 3.4.

3.3.2 Data Access

The links labeled as Fig 3.4-A and Fig 3.5-B point to almost identical pages that allow hub operators and village operators to upload content into the database. In the case of a hub operator, accessing the interface either through `localhost` or an Internet connection, the uploaded data is immediately visible in the hub database. In the case of a village operator, the uploaded data becomes visible in a hub or village "view" only later, after incurring some Postmanet delay. Much detail of the content uploaded, in the form of metadata, can be optionally specified at the time of the upload, which may aid subsequent searches of the database.

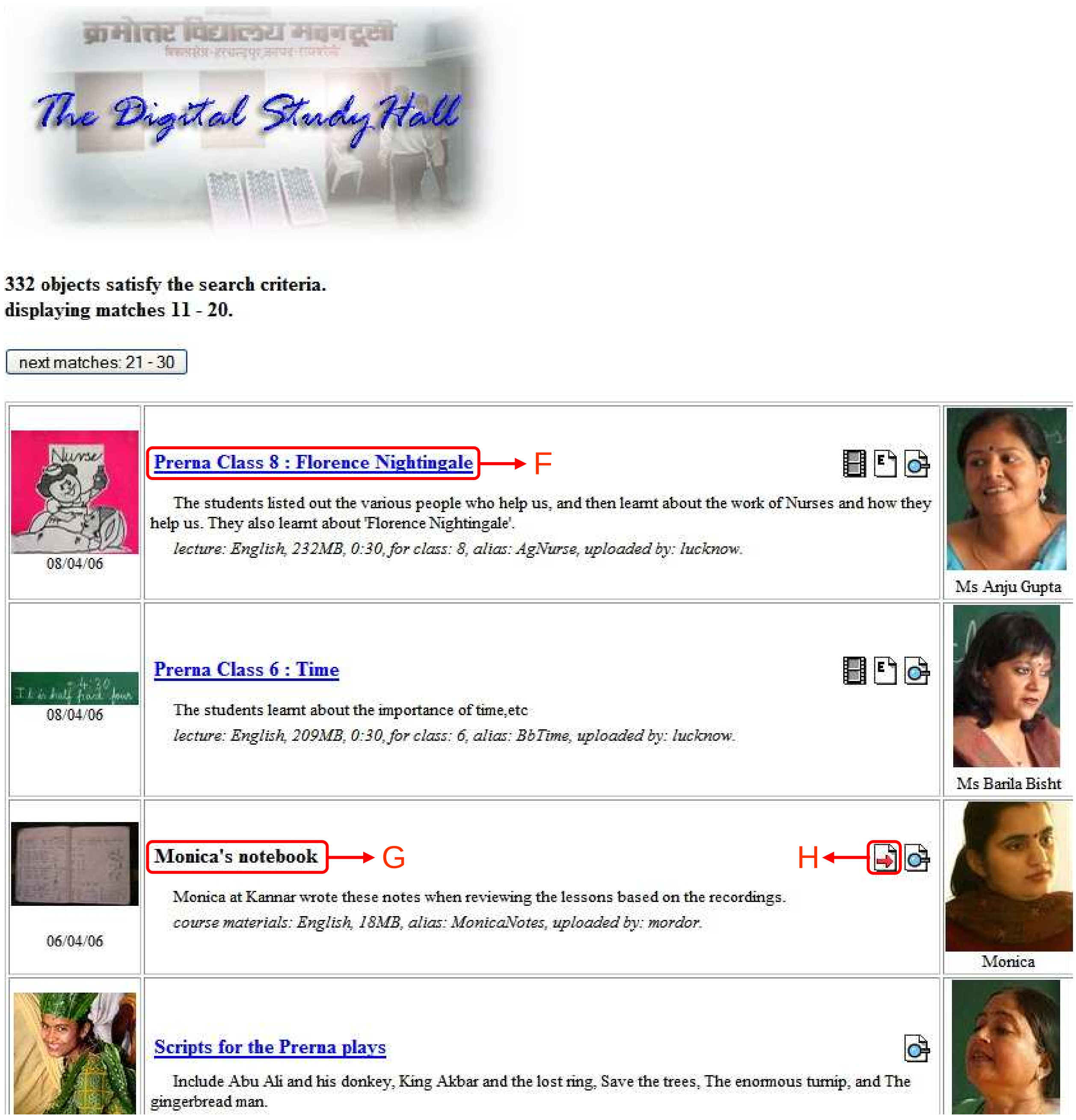
The links labeled as Fig 3.4-C and Fig 3.5-D lead to pages that allow operators to view the content in the database. Via the substantially similar versions of the browse, list, and search pages, a hub operator eventually reaches a content listing page, such as the



332 objects satisfy the search criteria.
displaying matches 11 - 20.

08/04/06	**Prerna Class 8 : Florence Nightingale** The students listed out the various people who help us, and then learnt about the work of Nurses and how they help us. They also learnt about 'Florence Nightingale'. *lecture: English, 232MB, 0:30, for class: 8, alias: AgNurse, uploaded by: lucknow.*	Ms Anju Gupta
08/04/06	**Prerna Class 6 : Time** The students learnt about the importance of time,etc *lecture: English, 209MB, 0:30, for class: 6, alias: BbTime, uploaded by: lucknow.*	Ms Barila Bisht
06/04/06	**Monica's notebook** Monica at Kannar wrote these notes when reviewing the lessons based on the recordings. *course materials: English, 18MB, alias: MonicaNotes, uploaded by: mordor.*	Monica
	Scripts for the Prerna plays Include Abu Ali and his donkey, King Akbar and the lost ring, Save the trees, The enormous turnip, and The gingerbread man. *course materials: English..., 5MB, alias: PlayScripts, uploaded by: mordor.*	

Figure 3.6: A hub content list page.



332 objects satisfy the search criteria.
displaying matches 11 - 20.

08/04/06	Prerna Class 8 : Florence Nightingale The students listed out the various people who help us, and then learnt about the work of Nurses and how they help us. They also learnt about 'Florence Nightingale'. *lecture: English, 232MB, 0:30, for class: 8, alias: AgNurse, uploaded by: lucknow.*	Ms Anju Gupta
08/04/06	Prerna Class 6 : Time The students learnt about the importance of time,etc *lecture: English, 209MB, 0:30, for class: 6, alias: BbTime, uploaded by: lucknow.*	Ms Barila Bisht
06/04/06	Monica's notebook Monica at Kannar wrote these notes when reviewing the lessons based on the recordings. *course materials: English, 18MB, alias: MonicaNotes, uploaded by: mordor.*	Monica
	Scripts for the Prerna plays Include Abu Ali and his donkey, King Akbar and the lost ring, Save the trees, The enormous turnip, and The gingerbread man.	

Figure 3.7: A village content list page.

one shown in Figure 3.6, and a village operator eventually reaches a content listing page, such as the one shown in Figure 3.7. A data link such as the one labeled as Fig 3.6-E or Fig 3.7-F leads to directly playable content.

These content listing pages are the first places where we see some subtle differences depending on who the operator is and where the operator is located. As an example, note the “Monica’s notebook” data link, Fig 3.7-G, is disabled on a village content listing page. In contrast, all the data links, on a hub content listing page, such as the one in Figure 3.6, are always active. This is because the village school equipment has only limited storage, so it behaves like a cache, where some data items could be missing. A hub site, on the other hand, always has all the data. We will discuss later how these “missing links” can be “filled in” as the village and hub sites communicate with each other, principally through the Postmanet. When an operator clicks a data link on a hub content listing page, he or she is also greeted with a window whose access is more or less restricted depending on whether the operator is accessing the hub database remotely via an Internet link or locally at the console.

These data access operations are the most frequently used actions. They are easy to use, and, despite some subtle differences, deliver a substantially similar experience regardless the location or the type of the network used by the operator.

3.3.3 Data Requests

We support both pull and push requests. A village operator can issue pull requests of the desired data that is missing from the village view of the database. For example, as mentioned above, Fig 3.7-G represents a missing data link. The village operator simply clicks the arrowed document icon (Fig 3.7-H) to record a pull request. At the end of the day, the village operator clicks the two links in the “Discs” section of the front page (Fig 3.5-I), which automatically generate the P-disks destined for the nearest hub.

Similarly, a hub operator can issue push requests of data destined to either villages or other hubs. For example, a hub operator, via either `localhost` or an Internet connection, reaches a hub site. After specifying which other site he/she desires to push data to, he/she may click a request icon (labeled as Fig 3.6-J) to record a push request. At the end of the day, a hub operator clicks the top two links in the “Discs” section of the hub front page (Fig 3.4-K), which automatically generate the P-disks destined for the intended receiver sites.

In our current system, a village site’s view of the database is entirely determined by the hub database it is attached to. A village database does not contain all the data of its hub; and the “missing links” can be filled by either pull from the village or push from the hub. (A village and the hub it is attached to observe the “inclusion property,” to borrow a caching jargon.) At the present, the hubs have independent views of their own databases. They exchange data with each other via mutual pull or push. (There is no inclusion property for the hub relationships.) The combined effect of this independence and sharing is that it allows the various NGOs running the hubs to retain complete freedom of their own databases without fear of unintended or uncontrolled propagation or replication. In the next generation system that we are building, there will be tighter coordination among the hubs.

3.3.4 Receiving Incoming P-disks

When a village operator receives an incoming P-disk, he/she inserts the P-disk into a DVD drive. As part of a DVD autorun program, a simple dialog box pops up on the

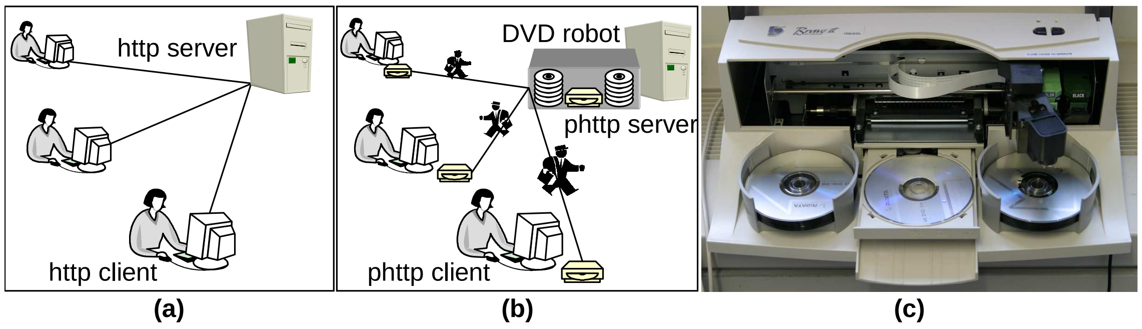


Figure 3.8: The use of a "DVD robot" as a part of a hub server. (a) A conventional web server and its clients. (b) A PHTTP server and its clients. (c) A DVD robot that is a key part of automating a PHTTP web server operations.

screen and asks the operator to press any key to confirm the desire to proceed. After a key is struck, the operator is asked to wait, while the system processes and integrates incoming data. When the process finishes, a village operator should see his/her view of the database refreshed. The database now should contain data that had been requested in earlier pull requests from the village and/or push requests from the hub. As we have discussed earlier, this process is designed to involve minimal manual intervention by the village operator.

The hub procedure for processing incoming P-disks is somewhat different, due to the fact that a hub serves multiple village spokes. When a hub operator receives incoming P-disks, he/she collects them in a stack, which is then placed in an input bin of a "DVD robot" (Figure 3.8). The discs from the village spokes and from fellow hubs can be freely mixed in the stack. The operator activates the robot by clicking the link Fig 3.4-L. The robot, in conjunction with the hub server, processes each P-disk and drops them in a stack in the output bin. During this process, there is no need of any operator intervention, and the process could run unsupervised overnight. When the operator arrives at the console the next morning, he/she would be greeted by an updated hub database view that includes data submitted by its spoke villages and fellow hubs. Other control information is also integrated from the incoming P-disks into the hub database view; examples include the merging of the pull requests sent via the P-disks and the push requests belonging to the same user on the hub machine, and system logs from the village machines. If the hub operator is satisfied that all the operations have terminated successfully, he/she may press the erase link (Fig 3.4-M), which instructs the robot to start the bulk erasure operation on the stack of the DVDs that have just been processed.

3.4 Content Production and Pedagogy

Obviously, building an effective distance learning system requires a lot more than making available a distributed database. Although the focus of this thesis is technical, we would like to briefly address the educational (or non-technical) aspects that are equally (if not more) critical to the success of the Digital StudyHall system in its goal of spreading high-quality education to resource-starved rural India. It is important to understand that the technical plumbing addressed in this thesis serves not to replace humans, but to amplify the reach and power of the limited human resources that we do have. A natural extension of the “whole systems” approach of the Digital StudyHall system is that it is foremost a “system” of people. The non-technical aspects also help us understand why traditional approaches, such as broadcast TV or satellite TV, are ill-suited for the educational purposes we seek.

Aware of the systemic difficulties one would face when attempting to build a system that aims to bridge the vast education gaps faced by schools and children of vastly different backgrounds, after several iterations of experimentations, we have developed a set of methodologies based on the understanding that in order to successfully exploit our technology platform, we must pay special attention to the following issues.

• *Contextually meaningful and coherent content.* We recruit the best teachers from the resource-rich middle-class schools to teach specially staged classes in front of an audience consisting of kids from neighboring slums. This is economically feasible because this is a one-time effort and the recorded content is to be reused at many other locales. This approach allows us to combine the best of the "good" and poor school environments: highly-skilled teachers in front of an audience whose background is similar to that of the rural audience. These staged lessons are systematic sequences designed based on the U.P. state board textbooks, and the teachers use the appropriate combination of Hindi and English for their classes. We also recruit advanced middle-class students in hub schools to participate in content production. These students are well-versed in the local language and culture and the resulting content ends up being highly relevant for the local context.

• *Active mediation by local teachers.* Obviously, playing the recorded videos alone in front of the village children, by itself, is not sufficient as an effective teaching vehicle. We require and train the local teachers (in the village and slum schools) to proactively engage their students while playing the pre-recorded videos. For example, when questions are asked, when board exercises are done, when role-playing occurs, when poems are recited, or when songs are sung on the video, the local teachers would pause the video and get their students to perform similar activities, so the class watching the video is as dynamic and as interactive. In a sense, the video and the local teacher form a team: the video provides a framework, an agenda, and a content and methodology model; while the local teacher supplies the crucial interactive component.

• *Continuous training, monitoring, and support for local teachers.* The local teacher training occurs in at least two ways. First, the best teachers at the headquarters school provide in-person demonstrations to the local teachers on how to actively engage their students when using the video content. Second, the local teachers study the videos on their own to improve their mastery of both the subject matters and the methodology in the specific contexts of each lesson. Unlike conventional training workshops that last only for a short period of time and can be too abstract, the kind of training a local teacher receives from the supplied videos is ongoing, continuous, and highly specific. As the local teachers gain know-how and confidence, they may choose to use less help from the videos during live classes and improvise more. Graduating the teachers in this way, in some sense, can have a bigger impact than graduating students. The hub staff periodically visits the village schools to provide in-person feedback. For the local teachers who have gained a large degree of independence, we provide additional supplementary materials (such as complementary digital stories) to allow them to be even more effective.

It is useful to compare the Digital StudyHall against satellite-based approaches. Satellite-based approaches are expensive and they require a great deal of support infrastructure. Satellites are a good broadcast medium: a small number of one-way streams consumed by a vast number of content consumers. But broadcast models are poor ways of delivering carefully crafted local content that is tailor-made for small regional audiences, and allowing two-way exchanges. Satellites can also be used to support non-broadcast or even two-way communication. If we do that, however, we face a severe bandwidth problem: each of a large number of communication channels only gets a small fraction of the aggregate bandwidth. The bandwidth limitation is especially serious on the uplinks. One important advantage of the Digital StudyHall is that it allows high-bandwidth, any-to-any, point-to-point communication, which in turn enables a high degree of content customization and rich two-way exchanges.

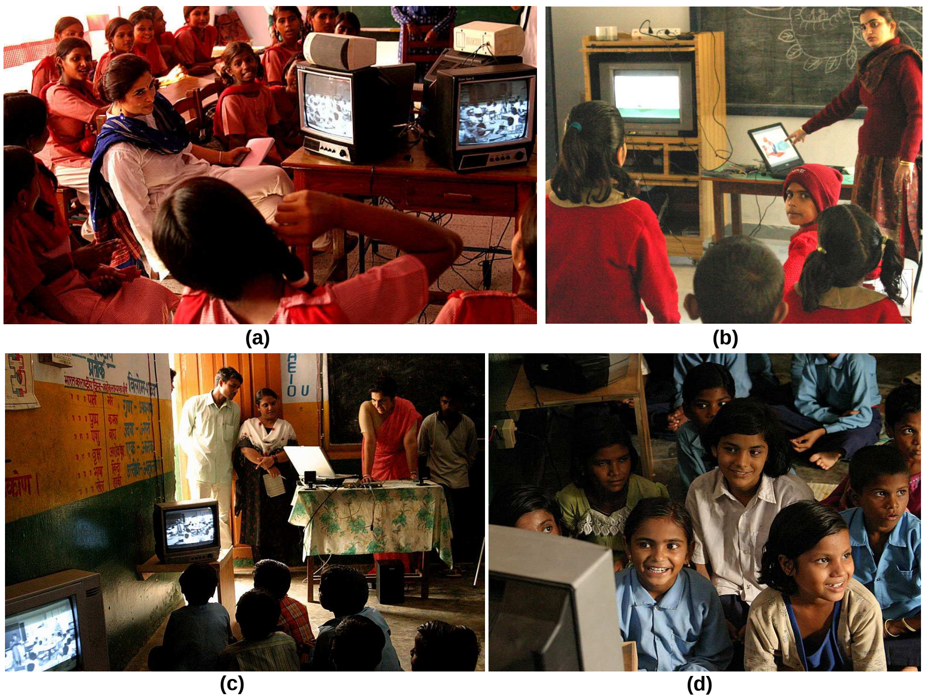


Figure 3.9: Testbed schools using the prototype. (a) In the Prerna school, a school for girls from Lucknow slums. (b) In the Kannar village school. (c) and (d) In the Madantoosi village school.

3.5 Status

A live deployment of a prototype has been in use by students starting in July of 2005. In the space of about eight months, a database at the Lucknow hub has accumulated about 70GB of content. This includes more than 170 high-quality MPEG4 recordings of lessons staged by the best teachers at the hub school. The remainder includes Hindi science courseware, digital stories, recordings of drama performances designed for various educational purposes, and training materials for local teachers, all of which have been produced by students and staff at the Lucknow hub school. As the high-quality content is quickly and cheaply generated, it is being continuously pushed out to two test village schools and a slum school (Figure 3.9). At the time of this writing, we are working with other educational NGOs and beginning to set up additional hubs in Calcutta, Pune, Bangalore, and Punjab. Preliminary results appear promising, and the system seems to be playing an effective but subtle role of blurring class differences in a highly stratified society. We hope to eventually scale up the system to cover a far greater number of villages and children, contributing toward the Millennium Development Goal [63] of universal primary education.

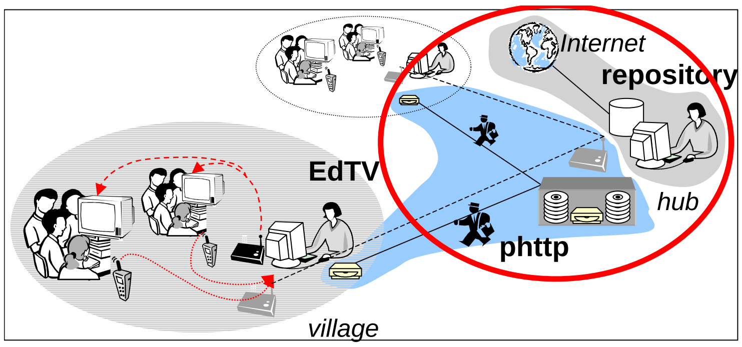
Chapter 4

PHTTP: A Browser-Based Substrate

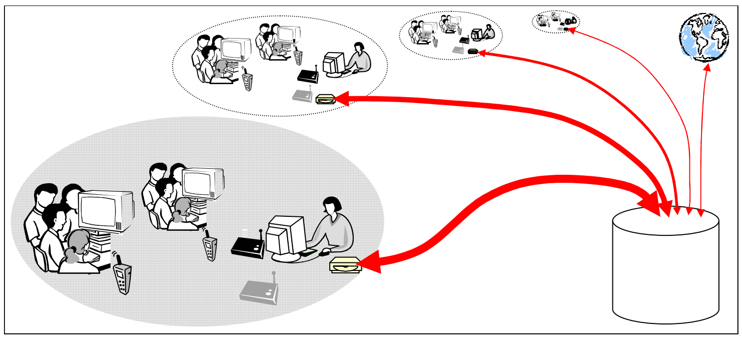
In the previous chapter, we have discussed the interface of the Digital StudyHall content repository, operating on top of a combination of the postal links and broadband Internet links. From this discussion, one may recognize that the utility of such a distributed content repository is useful for applications beyond the immediate application of the Digital StudyHall; so the systems support and benefits that we should generalize from this exercise should be greater than those of the mere transport layer. In this chapter, we discuss how a general purpose PHTTP layer manages *data*, *code*, and *devices* in support of browser-based Postmanet applications.

4.1 Overview

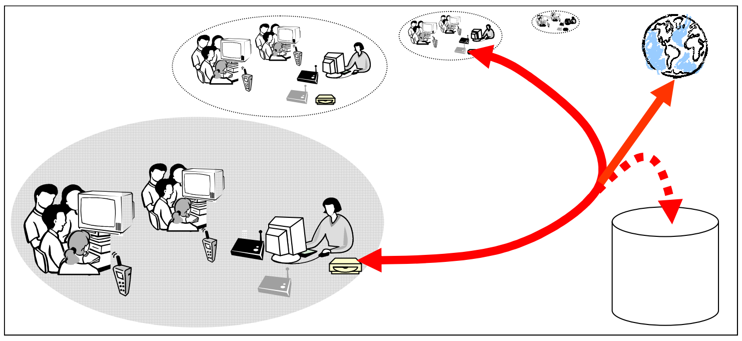
A distributed repository that is programmatically accessible by the postal system (Figure 4.1(a)) provides an abstraction that is akin to that of a distributed file system (Figure 4.1(b)): it makes a single name space available “everywhere,” including those places that lack conventional networking access, and allows read, write, navigation, search and



(a)



(b)



(c)

Figure 4.1: The synergy of a postal system-based "network" and a content repository. (a) A content repository that is accessible by both the conventional Internet and the postal system, where the postal system connects the Digital StudyHall villages to the hub repository. (b) Viewing the postal system-accessible repository as a common "storage system" that all sites can read from and write to. (c) Viewing the postal system-accessible repository as a "network with memory."

other primitives to be performed on this name space. This file system analogy means that the repository is a sufficiently general abstraction that can be used to support other applications. While this view is a file or storage system analogy, an alternative point of view of the synergy between the repository and the postal system is a network analogy (Figure 4.1(c)): any communication between any pair of nodes is “routed” through “the repository,” which keeps a copy of the content that is being exchanged. In other words, this is a “network with memory.” This is particularly appealing for applications such as the Digital StudyHall because it allows numerous content reuse opportunities.

The importance of including the repository in a Postmanet API is even better understood if we consider the difference between the Postmanet and sending cassettes in the postal system, which people have been doing for a long time. A cassette in the postal system is sent from one person to one other person; and one cannot really send a cassette to the “entire world.” In the Postmanet, when a P-disk’s content is deposited in the repository, which is accessible by the “entire network,” a person literally has the power to send something to the entire world. The power of a “network effect” is much more pronounced than making point-to-point connections between people who already know each other; it is about building eBay-like communities of people who otherwise would not have connected with each other. So the repository plays an important role in realizing the “network effect” of the Postmanet, and this is why the repository abstraction is a crucial part of the systems support we provide.

So far, the discussion above has focused on *data*, the issue of making a distributed data repository accessible everywhere. We also need to deal with *code*: in order to make the Postmanet applications work, it is not enough to make data easily accessible; we also need to put in a systematic mechanism to distribute code. In the simplest case in the conventional world, a browser client enlists code on a remote web server (such as `cgi-bin`) to process data that is either local to the web server or submitted by the browser client. For other cases, a browser client downloads and installs a plugin that can both process client data and communicate with the remote web server. In either case, these existing solutions break down without a conventional Internet connection. A Digital StudyHall village spoke, which may only be linked to the outside world via the postal link, needs to have the "correct" version of code in order for it to correctly function with respect to the rest of the "network." Furthermore, an entirely new Postmanet application may be distributed via the Postmanet along with its initial data to a node for it to bootstrap itself. What we need is a distribution mechanism in which data and code always accompany each other so a node always "knows" what to do with the data. In principle, this approach is not unlike previous "active networking" approaches, which transmit both data and code on wires [1, 12, 44, 69]. Almost all existing active networking efforts to date, however, have consciously avoided tackling persistent storage inside the network. The type and granularity of codes being distributed in a Postmanet are qualitatively different.

In the rest of this chapter, we discuss aspects of the PHTTP software, shown in Figure 4.2. The PHTTP layer consists of two major internal modules: the lower-level *transport* layer, and the higher level *repository* layer. Applications, such as DSH and Postmanet-based email, which desire the use of a distributed object repository managed by the PHTTP system, are built on top of the *repository interface*. Applications, such as a Postmanet-based backup software, which manage its own data store, are built on top of the lower-level *transport interface*. The transport module includes *device managers* that interact with hardware devices (such as DVD robots and regular DVD writers) that are responsible for reading/writing P-disks. We discuss the object repository API in Section 4.2.1 and the transport level API in Section 4.2.2. We then discuss the internals

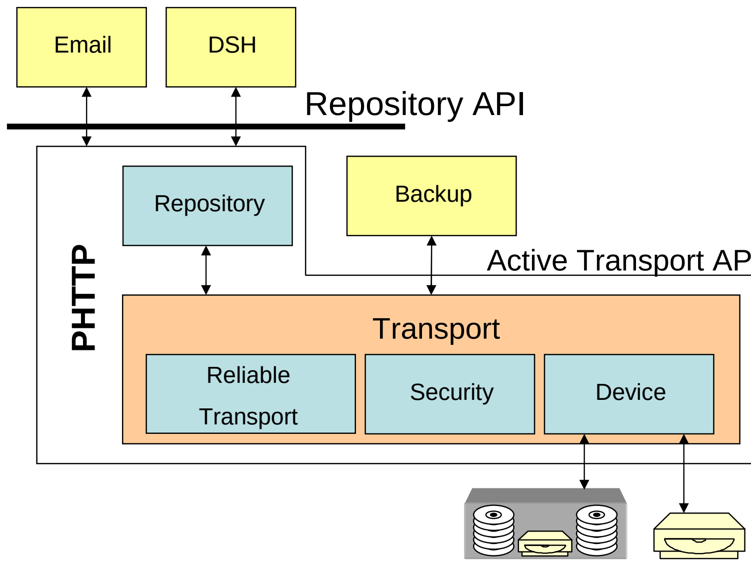


Figure 4.2: PHTTP: APIs and internal modules.

of the two modules: the transport level internals in Section 4.3 and the object repository internals in Section 4.4.

4.2 The PHTTP APIs

4.2.1 The Object Repository API

The object repository provides an API that largely mirrors the needs of the DSH application: much of what the hub and village human operators do is handed down to the repository API by thin layers of the hub and village wrappers of the DSH application. Any user (from either the hub or the village side) may upload, browse, and search for objects in the repository. A hub user may specify which objects to push to which remote Postmanet sites, which may be either village sites or fellow hub sites. A village user may specify which objects to pull into its local view from the hub that it is attached to. While the DSH application currently uses almost all of what the repository API has to offer (with the exception of versioning features), a different application may use only a subset of this interface. For example, an email client built on top of the repository API may not use the pull feature and rely entirely on server pushes.

Our experience with the DSH application has taught us much about what kind of support a Postmanet object repository needs to provide. The very nature of a Postmanet application is such that it needs to manage (1) big objects, and (2) many objects. A number of the object repository features are specifically designed to handle these requirements.

Due to the potentially large number of objects present in the object repository, we need good ways of quickly finding data. The principal way of accessing the object repository is searching on a "flat" name space. The overwhelming majority of the objects in the repository, however, are large multimedia objects (such as video, audio, and images), and the state of art of direct indexing of such objects is not yet satisfactory. So what is needed is good metadata that accompanies all data. Metadata management is an integral part of the object repository interface. Indeed, metadata is part of the object definition: each object has its metadata component. And much of the object repository interface specifically deals with metadata. For example, the "upload" operation of the object repository expects metadata to be a part of the call arguments. We will use the terms metadata and attributes interchangeably. The object repository layer has its set of "standard" attributes, such as upload time, size, and data type. An application that runs on top of the object repository may define its own additional attribute sets, such as teacher names, academic subject topics, applicable student ages, and the types of devices that are used to capture the content.

In addition to metadata, an additional mechanism for dealing with unwieldy large media files is through the use of the "helper files." These helper files may serve various purposes. For example, instead of downloading hundreds of megabytes of a data file through a broadband connection only to find out that this is not the content one is looking for, a fellow hub operator may quickly browse and examine some small “excerpt files” to help determine whether a data object in question is of interest. The object repository manages these helper files uninterpreted; and it is up to the application built on top of the object repository to interpret and use them in application-specific ways. The Digital StudyHall application, for example, uses these helper files to store thumbnail images (of content and authors), extra teaching materials and instructions for teachers.

These small files (metadata and helper files) contribute to the materials being indexed for searches. Generic key word searches are also provided as part of the object repository interface so the different applications layered on top do not have to reinvent their own. Application-specific searches (such as the “advanced search” links in Figure 3.4 and Figure 3.5) are implemented by applications (DSH, in this case).

Our conscious effort of coupling large data items with smaller informative files is also an integral part of the general Postmanet theme of exploiting dual networks: getting fresh, informative, and compact summary information easily and quickly on an LLLB link, while the bulk data travels on an HLHB link. Even for sites that have no access to an LLLB link today, these metadata and helper files are liberally disseminated via P-disks, even without explicit pull or push requests, to help present a coherent view across distributed Postmanet sites.

In addition to using search mechanisms to locate data, the object repository also allows an upper layer user to define its own hierarchical name space, which is “overlayed” on top of the flat name space, in the sense that the “leaf nodes” of the hierarchical name space are pointers to the “real” objects in the flat name space. An application constructed on top of the object repository may use the hierarchical name space to implement features such as a topic-based hierarchy, which a human user may easily browse and navigate. The DSH application currently selectively uses some of the metadata to automatically construct a hierarchy so a user may browse the content repository based on information such as the site that has created the content, the type, and the creation time of the content.

4.2.2 The Active Transport API

As we have discussed in the overview section of this chapter, distributing *data* is only half of the story; distributing *code* is equally important for “disconnected” sites to operate over the Postmanet. In our current design, we have made a decision of managing code distribution at an API lower than the object repository API discussed in the previous section—code transmission is done at the “active transport” layer (Figure 4.2), some of whose other responsibilities can probably best be summarized as “TCP on top of the postal system,” such as providing reliable transmission on top of unreliable P-disks instead of unreliable wires. Placing code distribution in a lower layer than the object repository interface allows the repository itself to be an “application” utilizing the code distribution mechanism to distribute, install, update the repository layer code. Other applications may choose not to use the object repository but still need to rely on the code distribution mechanism to bootstrap themselves.

A P-disk arriving at a village may contain data and code belonging to multiple applications, each of which occupies its own location on a P-disk. A known location inside each application’s space contains the entrance point of a piece of application-supplied “handler code.” When control is eventually transferred to each of these handlers, a handler typically serves two purposes: first, it installs (or updates) the application-specific code on the village machine; second, it copies data from the incoming P-disk to the village machine in an application-specific manner. (We refer to this handler as a `client_copyin()` call.) The use of application-specific handlers to accomplish these tasks provides a great deal of flexibility to the applications. In principle, this approach is similar to prior asynchronous communication mechanisms such as Active Messages [64] and the programming languages built on top of them [10], in which handler codes associated with messages are asynchronously executed upon arrival of the messages to incorporate the newly arriving data into ongoing computations. In all these cases, one of the benefits of the application-supplied handlers is that they can integrate incoming data in an intelligent fashion that can avoid the inefficiencies involved in hardwired data copiers.

While the above discussion concerns `client_copyin()`, the process of taking messages out of the transport vehicle and integrating them into applications, we also need the opposite `client_copyout()` call, the process of taking messages that applications desire to send, and placing them on transmission media, the P-disks. At the end of a day, prior to a postman visit, the village operator clicks a link to generate outgoing P-disks. This user command is sent to the active transport layer, which seeks out and runs a known entrance point function in each of the installed Postmanet applications. The responsibility of each of these application-specific `client_copyout()` functions is to package up "messages" sent by each application throughout the day to prepare them for writing to the outgoing P-disks. Note that this "packaging" does not necessarily require time-consuming data copying: for example, we may simply move data from application-managed message buffers to a staging area reserved for DVD images that are to be burned. Once these `client_copyout()` functions terminate, the act of actually generating the outgoing P-disks, which contain data from the multiple applications, is completed by the application-neutral transport layer.

(In the above discussion, the use of application-specific code occurs across the active transport API on the village side. Another occasion where higher-level application code is employed in lower-level Postmanet support software occurs across the object repository API: the village view of the object repository, behaving like a cache, which cannot hold all data at once and must evict some objects, uses an application-supplied function to calculate which objects to evict first. For example, the DSH application supplies a function to the object repository that calculates a "cache worthiness score" for each object based on information such as data size and when the village has received the data.)

The use of application-supplied handlers or code across API boundaries is an important approach to defining Postmanet software APIs. It allows relatively simple lower-level softwares to accommodate sophisticated or unanticipated higher-level functionalities in an environment, where the lower-level softwares on a "disconnected" machine cannot be easily updated in an ad hoc fashion.

The discussion above has focused on village side (or PHTTP client) operations. Similar handlers are at play in the hub side (or PHTTP server) operations. An application built on top of the active transport layer is responsible for supplying a pair of handler functions: `server_copyin()` and `server_copyout()`. The former processes and copies data from incoming P-disks into application space; and the latter stages outgoing data from application space into outgoing P-disk image areas.

In summary, a Postmanet application running on top of the active transport layer is therefore constructed in two pieces: one part runs on the PHTTP server side, and the other runs on the PHTTP client side. The server component contains the two entrance calls: `server_copyin()` and `server_copyout()`. The client component contains the two entrance calls: `client_copyin()` and `client_copyout()`. Both the server and client components are stored on the server side. Part of the responsibility of the `server_copyout()` call is to place a copy of the client component on outgoing P-disks, and part of the responsibility of the `client_copyin()` call is to install the client component on the destination client machine before any other actions are to take place on a PHTTP client machine. Any code modification always originates on the PHTTP server, and is automatically propagated to any PHTTP client machine it communicates with.

4.3 Transport Layer Internals

Having discussed the APIs top-down (from the object repository API to the active transport API), we now examine the internals of these modules bottom-up (Figure 4.2), starting with the P-disk device manager.

4.3.1 Device Manager

The job of the device manager is to handle a variety of hardware devices that deal with the P-disks used in the system. (Our experiences have also shown us small modifications to the hardware or firmware of devices that could improve the PHTTP operations.)

We use DVD robots to automate the generation of outgoing P-disks and processing of incoming P-disks. In addition to their large capacity, low cost, and small weight, the reasons that have compelled us to choose DVD media include the fact that we can use commercially available (and cheap) robotic arm-operated DVD processors, a crucial part of our automation strategy. Subject to available hard disk buffer space, an arbitrary number of outgoing P-disk images can be prepared, and the burn commands are queued for the robot controller. Other Postmanet operations can continue in parallel while the robot works on the queue in the background. Similar background parallelism can be exploited for processing a stack of incoming discs.

The “Primera Bravo” line of “DVD publishers” (Figure 3.8(c)) that we use were originally designed for very different purposes in mind. The factory-supplied GUI allows operations such as burning many identical copies of the same disc image, or duplicating a stack of existing DVDs. They were not designed to support programmatically controlled production or processing of unique discs. For our purpose, we needed to gain access to an unpublicized internal interface, which was poorly engineered and poorly documented by the manufacturer, and required quite a bit of reverse-engineering to understand its behavior under various error conditions. A robot interface that allowed complete programmatic control would have made our task much easier. Another major defect of the robot interface is that it does not provide good support for *pointers* to data, and as a result, significant processing time is wasted on data copying.

We do not require all hub sites to use a DVD robot: a hub connected to fewer spokes may not need one. The device manager ensures that it presents the same interface to the rest of the software stack, regardless what underlying hardware device is present. All P-disks are created with an `autorun` file on them. When an incoming P-disk arrives at a site that does not use a robot, regardless whether it is a village or a hub site, the `autorun` file ensures that an operator can easily initiate processing of an incoming P-disk by simply inserting the disc in a drive and striking a key for confirmation. We exercise care in the software stack to synchronize access to shared data structures, so that a powerful hub machine can be connected to multiple DVD drives, all of which could be processing DVDs in parallel.

Some of our village spokes do not use full-fledged computers; instead, they use specialized MPEG-4 DVD players. We use a uniform DVD disc format that, in addition to being accepted by the transport layer, ensures that the MPEG-4 content on the discs can be easily navigated and played on these DVD players by a human operator as well.

4.3.2 Transport Data Security

There are mainly three types of security problems that we may need to address: (1) spoofing, the problem of an attacker impersonating a different sender, (2) tampering, the problem of an attacker intercepting and modifying data sent by others, and (3) eavesdropping, the problem of an attacker reading communication that should have been private. If no action is taken to guard against these problems, all of them can easily occur to DVDs transmitted in the postal system, perhaps more easily than what can happen to packets on conventional networks. The solutions, however, are essentially no different from those employed on more conventional networks.

In our current design, we have decided to address problems (1) and (2) at the transport layer. Like what one would do for wired networks, the solution is through signing P-disk data with digital certificates by senders. Correspondingly, the receiver needs to use the sender's key to verify the authenticity of data on an incoming P-disk. Such keys (along with postal addresses) of Postmanet sites, from which one receives P-disks, can be obtained via one of several possible ways: (1) via an LLLB Internet, if it is available, (2) from a well known trusted authority via any communication mechanism, or (3) via other "channels" such as paying a personal visit to the physical site of the sender.

We have decided to "punt" problem (3), the eavesdropping problem, to layers above the transport layer, since this is not necessarily a problem that all applications want addressed for them by the transport layer. This decision is analogous to the fact that SSL is layered on top of TCP and is only used by applications that choose to use it. In the case of code layered directly on top of the active transport API, an application that desires keeping its data private would place encryption code in each of the two "halves" of its code, the PHTTP client component and the PHTTP server component (as discussed in subsection 4.2.2), ensuring that all application data on P-disks are encrypted. In the case of code layered on top of the object repository API, an application may place encrypted objects in the object repository.

4.3.3 Isolation of Application-Supplied Code

Recall that we use application-supplied installer and data copiers during the processing and generation of P-disks, in the form of the two pairs of functions expected by the active transport layer: `client_copyin()`, `client_copyout()`, `server_copyin()`, and `server_copyout()`. We need to ensure that these handlers do not interfere with or damage each other or the rest of the system.

One simple way of insulating the handlers from each other is to run them as individual processes with their own separate address spaces. Access to persistent storage also needs to be restricted so each of these processes can only read and write certain directories. More efficient alternatives than the process model, such as software-based fault isolation [65] and safe language-based extensions [8], also exist. All other resources on a node may need to be accounted for as well [6] .

In addition to the transport components on the different machines, the application components (the PHTTP client and server components) also may desire to authenticate each other. Existing cryptographic techniques for authentication, secure booting, and secure links can be used for this purpose [70, 18]. None of the mechanisms discussed in this subsection (4.3.3), however, has been implemented, and they are part of our future work.

4.3.4 Reliable Transport

The type of mishaps that can happen to conventional network packets can happen to P-disks in the postal system. The discs can get lost, damaged, or delivered out of order. And if retransmission is done, a receiver may see duplicates of transmitted data. These are precisely the issues that a conventional transport layer protocol (such as TCP) deals with. The basic approach to solving these problem is similar to what is employed for conventional networks: keeping state at the communication end points, which is used for managing acknowledgements, retransmissions, and suppression of duplicates. What is different, in the context of the Postmanet, is that the amount of data and the durations of the delays involved are far greater than those seen in conventional networks, so we need to adapt or modify the existing approaches.

The granularity at which we manage reliable transport is P-disks, each of which is identified by a P-disk ID. Each sender saves the P-disk images of the recently sent discs in case they need to be retransmitted. Acknowledgements are piggybacked on P-disks traveling in the opposite directions. P-disk images whose receipts are confirmed by acknowledgements are deleted; so are the images that have reached a certain age threshold. (On a village PHTTP client machine, which has limited disk space, we currently only keep a “buffer ring” of five old outgoing P-disk images: when we are about to save a sixth outgoing P-disk image, we delete the first. Hub PHTTP server machines have more disk space and we keep a deeper buffer space.) In our current practice, retransmissions are explicitly initiated by human operators when they are alerted to the presence of disc images that have not been acknowledged for a long period of time. Each site also maintains a list of IDs of P-disks that have been received, processed, and acknowledged, so potential duplicates that arrive in the future can be safely ignored.

Currently, the PHTTP transport layer does not enforce in-order delivery. The object repository built on top of the transport layer uses its own timestamps to ensure that older versions of object repository data do not accidentally overwrite newer versions. Except for this issue, the object repository communication can be considered idempotent, so out-of-order delivery of object repository P-disks does not cause problems. Also, in addition to utilizing the transport layer retransmission mechanism, the object repository layer is also capable of saving the request operations that have been used to generate outgoing P-disks. These requests are themselves stored in the object repository as new "objects." They can be easily accessed via search and browse operations built into the object repository and used to regenerate outgoing P-disks for retransmission.

A retransmission approach that is probably unique in the Postmanet is *proactive retransmission*: a P-disk sent on a later day could be used to store a copy of the data that was sent on a previous P-disk on an earlier day, if the later P-disk has extra space to spare. If the earlier P-disk is lost or damaged, the duplicate on the later P-disk allows the receiver to successfully receive the original without having to potentially wait for a long "time-out" and "reactive retransmission." This is an example of the recurring Postmanet theme of exploiting its plentiful resources (excess capacity, in this case) to compensate for its handicaps (long delay, in this case). This approach is on our to-do list of future implementation enhancements.

4.4 Putting It All Together: Implementation of the Object Repository

Having discussed the internals of the lower-level transport layer, we now examine the implementation of the object repository operations.

Since the PHTTP applications are meant to be browser-based, they are structured as a collection of `cgi-bin` scripts hosted by the Apache web server. More specifically, recall that the object repository is built on top of the active transport layer, which requires its application to be structured as a PHTTP server component and a PHTTP client component; so the object repository code is structured as two parts, each of which is a collection of `cgi-bin` scripts, whose service is accessed via a browser interacting with the local Apache server. The PHTTP software is written in the Python programming language, and it totals about 10,000 lines of code.

4.4.1 Object Repository Internal Data Structures

• *Object store.* The key data structure is the local object store, which contains all data and metadata from the local machine's point of view. Each object is identifiable via an object ID (including a version number), which is guaranteed to be unique across all time and space. Each object consists of three parts: the data part can contain an arbitrary collection of files, organized in a hierarchy; the metadata part contains all the generic and application-specific attributes, stored as a Python "dictionary" (which is a Python term for an associative array); and the helper data part contains an arbitrary hierarchy of small files. At least the metadata parts of the object store could be stored in a conventional relational database. Currently, however, we use a file system-based implementation to store the entire object store. The accesses to the object store are such that they do not require the full transactional semantics. We do exercise sufficient care to ensure that the object store implementation is robust in the presence of crashes.

• *Trimmed object store.* A "trimmed object store" is a copy of an object store minus all the data components whose sizes exceed a certain threshold size. Other than the excluded big data objects, a trimmed object store includes everything a regular object store has, including all the metadata and helper files.

• *Search file.* All the metadata of all objects in the object store is reformatted to form the search file, which can be stored entirely in main memory and allows very efficient searches. The search file is also sent to other sites, offering a summary of all the data present on the sender host.

• *Hierarchical view.* This is a user-defined hierarchical name space super-imposed on top of the object store that allows a user to easily browse and navigate the database.

• *Request queues.* The pull and push requests for objects destined for any PHTTP site are maintained in a request queue. Each queue is stored as a Python "set."

We consider the object store to be the "ultimate truth," in the sense that all other data structures exist for either convenience/optimization or transient reasons, and are thus expendable. The search file can be regenerated from the metadata stored in the object store. Where an object is attached in the hierarchical view is stored as part of the metadata in the object repository; so the hierarchical view can be regenerated. The request queues can be stored inside the object store as regular objects, so an operator or a program can potentially "revive" and act on earlier requests.

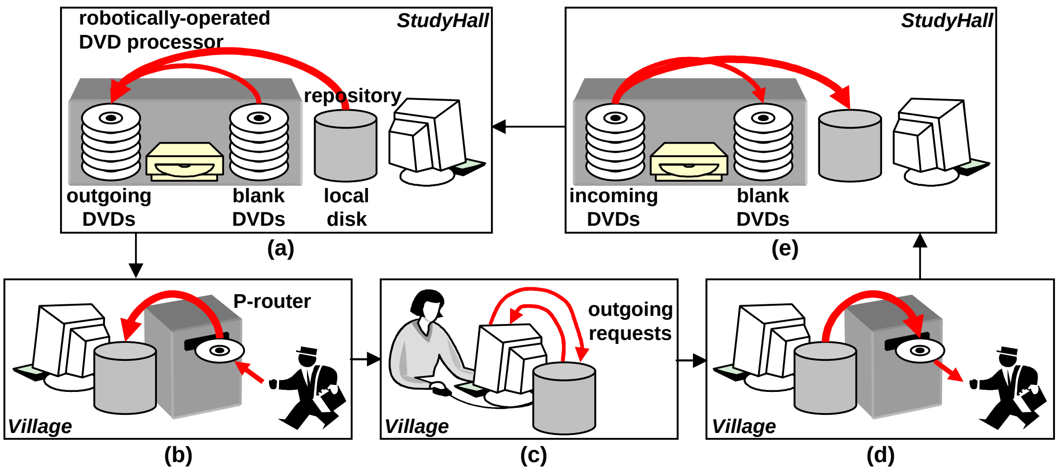


Figure 4.3: One cycle of PHTTP operations. (a) The server publishes P-disks for clients. (b) The village client incorporates data and code from the received P-disk. (c) A village operator interacts with the local client component. (d) The village client produces the P-disk to send back to the server. (e) The server processes P-disks received from the clients.

4.4.2 Implementation of the Object Repository Operations

We now examine a cycle of the PHTTP client-server operations and discuss how these operations are implemented. In the following discussion, when we refer to a "hub user" or a "village user," we refer to an application built on top of the object repository, or a human operator, or some combination of both.

Server Operations: Send

At the beginning, a hub user uploads data into the object repository, specifying metadata and hierarchical view paths in the process. The search file and hierarchical view are updated, allowing hub operators to easily search and/or browse the database. A hub user may specify which objects to push to which other sites. All these requests are recorded in the request queues. The hub operations may be performed at the console of the hub machine or remotely via a conventional Internet connection.

When a hub operator is satisfied that all the necessary data to be sent has been specified, he/she clicks a link to produce the outgoing P-disks (Figure 4.3(a)). In response, the lower-level transport software locates and invokes a `server_copyout()` entrance point in the object repository layer. For each P-disk, destined for a particular remote site, the `server_copyout()` function places the following items in its disc image: (1) objects specified by a request queue, copied from the object store, (2) a copy of the trimmed object store, (3) a copy of the search file, (4) a copy of the entire hierarchical view, and (5) a copy of the PHTTP client component of the object repository code. When the application-specific `server_copyout()` functions terminate, control returns to the transport layer, which continues with the remaining application-neutral tasks, such as placing `autorun` files in the disc images, and signing with digital certificates, before activating the DVD robot, which automatically produces a stack of finished outgoing P-disks.

Client Operations: Receive

When a P-disk arrives at a PHTTP client in a village, it is inserted into a drive (Figure 4.3(b)), and the simple generic `autorun` program on the P-disk starts executing. It locates the active transport layer code on the village machine that is responsible for processing incoming P-disks and execution continues there. This process is the equivalent of an "interrupt" delivered to a conventional operating system upon the arrival of a network packet. The transport layer code first performs some application-neutral tasks, such as verifying the authenticity and integrity of the incoming P-disk content. Then it locates an application-specific `client_copyin()` entrance point for each of the applications that have some presence on the incoming P-disk and runs it. This process is the equivalent of locating and executing individual “interrupt handlers” in a conventional operating system.

As the `client_copyin()` function belonging to the object repository application starts execution, it copies various repository-specific data and code. As it does so, it exercises care, by checking timestamps, to ensure that older versions of the data structures do not inadvertently overwrite newer versions due to out-of-order delivery of P-disks. The `client_copyin()` function updates the PHTTP client code on the village machine. It uses the trimmed object store stored on the P-disk to update the object store on the village machine: not necessarily every object represented in the village machine object store has its large data part present; but all the metadata is always present. The `client_copyin()` function copies the actual data objects pushed by the hub machine from the incoming P-disk into the village machine object repository. The `client_copyin()` function also updates the search file and the hierarchical view on the village machine. When the `client_copyin()` function terminates, the view of the object repository on the village machine is “synchronized” with respect to that of the hub machine at the time when the P-disk was produced by the hub machine. Control is then transfered back to the transport layer, which performs the followup operations such as recording a disc receipt acknowledgement, and optionally erasing the incoming rewritable disc, in preparation for later reuse.

At a later time, a human operator at the village interacts with a browser-based application that runs on top of the object repository (Figure 4.3(c)). All the services are provided by the PHTTP client component of the code that has been copied from an earlier incoming P-disk to its own location in the `cgi-bin` directory of the local Apache server, and the local object repository supplies the data. Repository data structures, such as the search file and the hierarchical view, “mirrored” from the hub server, enable the village client software to present a database view that is substantially similar to that seen at the hub server.

Client Operations: Send

During the interaction, the local object repository may receive two types of requests: “download” requests for data that is not present in the local object store but the village would like to receive, and “upload” requests of data that the village user want submitted to the object repository. The download requests are recorded in a request queue, and the upload requests are stored in a temporary staging area.

Prior to a postman’s visit, a village operator clicks a link to produce an outgoing P-disk destined for the hub site (Figure 4.3(d)). In response, as is the case on the hub server machine, the lower-level transport software seeks out and invokes a `client_copyout()` function in the object repository layer. This function places the following items in the disc image: (1) the objects that the village user want submitted to the hub, moved from their temporary staging area, (2) the request queue containing the object IDs of the objects that the village user would like to receive at the village in the future, and (3) system logs of the object repository software, which would allow hub operators to monitor and/or debug the health of the village software stack. When control returns to the transport layer, it performs the final tasks including placing in the disc image the acknowledgements of the P-disks already received at the village, adding another `autorun` file, digitally signing the content, and activating a DVD writer.

Server Operations: Receive

When the P-disks arrive at the hub from various client villages (or peer hubs), the hub operator collects them in a stack, and places them in one bin of the DVD robot. Then the operator clicks a link on the hub interface and leaves (Figure 4.3(e)). The transport layer activates the robot to load each P-disk into the drive and to process each in turn. (When the robot is used, the `autorun` files on the incoming P-disks are ignored.) The transport layer verifies the authenticity and integrity of the incoming P-disk. It inspects the acknowledgements on the incoming P-disk to decide which old disc images, saved earlier for possible retransmission, to delete. Then the transport layer turns control to the object repository-specific `server_copyin()` function, which performs the following repository-specific tasks.

For the objects uploaded by the village user via the P-disk, the `server_copyin()` function places them in the hub object repository. Again, it uses timestamps to rule out accidental overwrites. For the data requests in the request queue that specifies which objects that the village desires to receive, the function performs the union operation between this incoming request queue and the request queue for the same village user already present on the hub machine. The union is stored back on the hub machine and will be used to select outgoing objects for a future outgoing P-disk. For the village system logs on the P-disk, the function treats them just like regular objects submitted by the village—they are placed in the hub object repository. A hub operator may later mine the logs to look for potential problems with the village machine and potentially take preemptive actions. When the application-specific `server_copyin()` functions terminate, control returns to the transport layer, which records an acknowledgement for this new incoming disc, and optionally erases the incoming rewritable disc for later reuse.

At the end of this cycle, the village user shares a view of the hub repository, and the hub user has gotten the submitted data and requests sent by the village user. Now the cycle can repeat.

4.5 Summary

In this chapter, we have discussed the APIs and the internals of the PHTTP substrate, a number of software components that provide support for browser-based Postmanet applications. The types of support provided by the substrate include: (1) a distributed data (and metadata) repository, whose APIs allow search, browse, pull, and push, among other operations; (2) an API and a mechanism for splitting up and distributing application code that is designed to run on the Postmanet; (3) interfacing with hardware devices, such as the DVD robot; (4) best-effort reliable delivery through the use of acknowledgements and retransmissions; and (5) authentication and integrity checks of communication. The likely lack of conventional connectivity rules out more traditional ad hoc management of these issues: for example, one does not have the luxury in a “disconnected” village to “get online” and “download” the code he lacks. Everything the system needs, including data, code, control information, must be carefully packaged on and transmitted via P-disks in a systematic fashion to ensure that the entire distributed system functions correctly. This is why systems support, in the form of a PHTTP-like substrate, is critical if we want to realize the potential of the Postmanet approach.

Chapter 5

Routing

The PHTTP software stack discussed in the last chapter concerns systems support at the communication end points. In this chapter, we consider what happens *inside* the "network." More specifically, we now discuss how a P-disk is routed from the sender to the receiver. One may naively think that the postal system would take care of it all, but as we shall see, the issues in the Postmanet can be more complicated, especially if there are a large number of communicating parties.

So far, we have only had practical experience with a single hub-and-spokes testbed in the Digital StudyHall project. An important question is how we intend to scale up the system. In this chapter, we examine just one aspect of scaling up: the issue of routing. In the preceding chapters, we have mentioned how we intend to scale up the system by adopting a multi-hub-based approach: a network of hubs and spokes. Our assumption has been that the villages only communicate directly with the hub, and the villages do not communicate directly with each other. Even in such a multi-hub network, we still have some important questions to address about a single-hub network. For example, how many village sites can a single hub serve? Should we allow villages to communicate directly with each other without an intermediate hop through the hub? Will that approach help improve the scalability of a single hub? By how much? How can this approach of village-to-village direct communication be integrated with a routing infrastructure that employs multiple data distribution centers (P-centers)? Even within a single-hub network, to scale up, we may need to "plant" additional P-centers that help route data but are not full-fledged hubs that also specialize in content production and pedagogy research. In this chapter, we explore the various infrastructure-based and infrastructure-less routing approaches that a Postmanet can take, with specific emphasis on the implications of the "peer-to-peer" infrastructure-less routing strategies.

5.1 Routing Strategies

We start by considering the Netflix.com movie DVD rental business as an analogy. In the early days of the company, when customers returned their DVDs, they were always first shipped back to the company headquarters in San Jose. The disadvantage of this approach is obvious. For example, when an east coast customer A returns a DVD that A's next door neighbor B is waiting for, despite the close proximity of A and B, the DVD is forced to make a long detour through the west coast.

Nowadays, Netflix maintains multiple distribution centers throughout the country. A returning DVD is sent to the distribution center that is closest to the customer's home address. Upon the arrival of the DVD, the staff at this distribution center looks up a database of waiting lists to see if any other customer is waiting for this DVD. If there is, the DVD is then forwarded immediately to the waiting customer at the head of the queue. This improvement has significantly reduced the wait time for Netflix customers and unnecessary extra costs associated with long hops in the postal system. This analogy

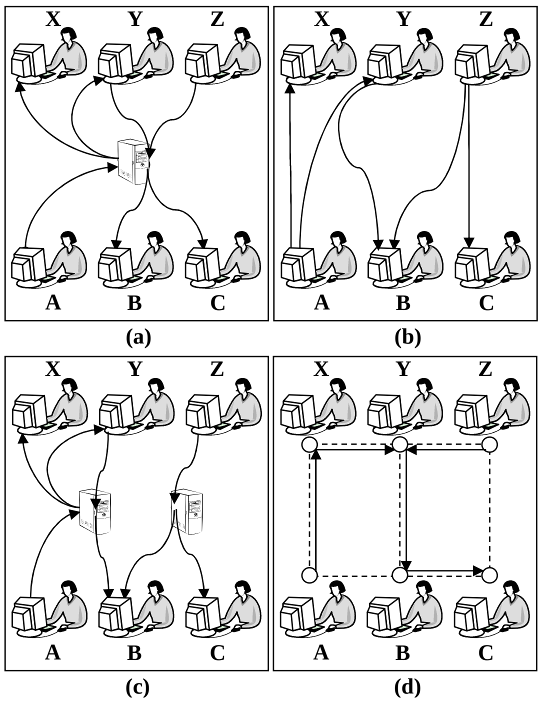


Figure 5.1: Routing strategies. A solid arrow denotes a single P-disk carried by the Postmanet on one postal hop. A dashed line between a pair of nodes in (d) denotes that it is permissible for these two nodes to exchange P-disks directly with each other. In all four panes, A sends different data items to X and Y, Y sends some other data to B, and Z sends different data items to B and C. (a) Centralized data routing via a single data distribution center. (b) Direct peer-to-peer data routing. (c) Data routing via multiple data distribution centers. (d) Indirect peer-to-peer routing.

is illustrative, but the issues in the Postmanet are even more complex. The data granularity and communication pattern in a movie rental business are much simpler than those seen in a generic communication mechanism. For example, data on movie DVDs cannot be split up, recombined, and placed on other DVDs.

In the following discussion, when we say a site "handles" *k* P-disks, we mean that the site may receive up to *k* P-disks and send up to *k* P-disks per postman visit; and when we refer to a "latency" metric, unless explicitly noted, it is in terms of the number of postal system forwarding hops visible to Postmanet participants. We consider the routing strategies illustrated in Figure 5.1. In the centralized alternative illustrated in Figure 5.1(a), an end user always sends/receives P-disks directly to/from a single data distribution center (called a *P-center*). Although any centralized solutions have obvious disadvantages, an important advantage of this approach is that each end user handles only a single P-disk, regardless how many other sites he communicates with per postman visit: as the P-center copies data from its incoming P-disks to its outgoing P-disks, it first demultiplexes incoming data and then re-multiplexes outgoing data, minimizing the number of P-disks handled in both directions. (Inexpensive robotic arm-operated, multi-drive DVD writers that can generate about 600 DVDs per day already exist today and they can keep manual labor cost to a minimum.)

In the direct peer-to-peer routing alternative illustrated in Figure 5.1(b), each user may need to prepare multiple P-disks for transmission, each of which destined for a different intended receiver. This approach has potentially better latency and lower infrastructure cost than that seen in Figure 5.1(a), but it may result in each site having to handle many P-disks. In a large scale peer-to-peer file sharing application, for example, the large number of P-disks handled per site could become a severe administrative and cost burden. This is an instance where the answer of "leaving routing to the postal system" is insufficient.

In the multiple-P-center approach illustrated in Figure 5.1(c), the geographically distributed P-centers allow some degree of geographical awareness in routing decisions, thus achieving latencies that are potentially better than those in Figure 5.1(a), but worse than those in Figure 5.1(b). In this example, we loosen the restriction that an end node is exclusively attached to only one P-center, so an end node can send/receive data to/from multiple P-centers. The number of P-disks handled per site depends on factors including the number of P-centers and whether P-centers communicate with each other. The advantages of this approach do not come for free, however, as the P-centers may require a substantial infrastructure investment. It is also possible to allow the coexistence of the alternatives illustrated in Figures 5.1(b) and (c).

In the indirect peer-to-peer routing alternative illustrated in Figure 5.1(d), a P-disk arriving at a site may contain data destined for other sites so, in some sense, the data copying tasks of a P-center is now distributed among the peer participating sites. In Figure 5.1(d), for example, a P-disk traveling on the $Z \to Y \to B \to C$ route delivers data sent by Y and Z to B and C. Using an analogy, one may view the P-disks as buses and messages as bus passengers: a passenger may need to switch buses to get from its source to its destination. If bus schedules are carefully planned and used, one may be able to limit the number of P-disks handled per site while still achieving good message latencies. An important advantage of this approach is that it does not require a P-center infrastructure.

A potential complication facing any peer-to-peer system is coping with misbehaving participants: a Postmanet user, for example, may fail to promptly forward data destined for his peers, alter or damage data, or read data that he is not supposed to. Routing protocols designed to deal with Byzantine faults [5] use a combination of techniques, including participant monitoring, destination acknowledgements, fault announcements, checksumming and encryption of data, authentication, fault knowledge sharing, and isolating faulty nodes. These Byzantine-tolerant protocols are directly applicable here and they can be integrated with a (suitably modified) Netflix-like service model, in which customers stop receiving additional service if they do not return outstanding discs already in their possession. Proactive data replication on multiple outgoing P-disks along different routes can further improve robustness and performance. In the context of a project like the Digital StudyHall, where the participants are collaborating on a common goal, the misbehaving peer issue might be of even less concern.

In Figure 5.1 Option (a) is a special case of option (c); and option (b) can be seen as a special case of option (d). If we can afford it, a properly provisioned infrastructure in terms of a number of geographically distributed P-centers (option (c)) should give the best quality of service. Ideally, the P-centers should be integrated into the existing postal system (or its rough equivalent, such as UPS or FedEx) so that some or all of the post offices themselves serve as P-centers, further minimizing delivery latency. Without relying on a P-center infrastructure, the peer-to-peer model (option (d)) may be the quickest way of incrementally starting the deployment of a Postmanet. It is also possible to mix options (c) and (d). The true value of the peer-to-peer approach may lie in its complementary nature, that it can co-exist with and improve upon the other infrastructure-based approaches. It is this model that we focus on first; and we examine later how P-centers can be integrated into this model.

5.2 Problem Definitions

• *Static routing graphs.* In Figure 5.1(d), suppose each user is only allowed to directly exchange P-disks with “neighbors” along the dashed lines. By constraining the number of such neighbors for each node, we limit the number of P-disks handled per site. A natural question is how such neighbors are chosen. In graph theoretic terms, the problem of simultaneously limiting the number of P-disks handled per node and maximum latency can be seen as that of constructing a directed graph with a large number of nodes while keeping the diameter and the maximum node degree small. The diameter corresponds to the maximum latency, and the degree of a node corresponds to the number of P-disks it handles. Although the problem of constraining both graph degree and diameter is applicable to general networks, we shall see that the quantitative tradeoff involved in the Postmanet (between postal system delays and the number of P-disks handled), and the need of generalizing the problem dynamically present unique challenges.

• *Dynamic routing.* The problem posed above concerns a static topology: a Postmanet node may directly exchange P-disks only with a small number of pre-determined neighbors. These static constraints may be unnecessarily restrictive. For example, in Figure 5.1(d), if *C* desires to send data to *A*, its data would normally be routed through *B*. But, there is no reason why *C* should not be allowed to send a P-disk *directly* to *A* if, on a given day, it does not overburden either of them. The question concerning a more dynamic approach is how to allow for such routing flexibilities without causing problems such as too many P-disks being handled by any one node on any given day. This is a routing optimization problem unique to the Postmanet.

• *Disseminating routing information and coordinating routing actions.* The questions are: (1) how is the traffic information (in terms of who is sending to whom) gathered? (2) who computes the routes? and (3) how are the computed routes disseminated?

• *Geographic awareness.* Obviously, not all postal hops are equal in terms of their geographic distances and postal delays. The question is how to construct routing graphs that can account for these factors.

• *Integrating P-centers.* We would like to understand how to best integrate P-centers into our routing mechanism, incrementally if necessary, to improve service quality.

5.3 Solutions to the Routing Problems

We now answer each of the questions posed in the last section.

5.3.1 Static Routing Graphs

Although dynamic routing should undoubtedly out-perform the static approach, especially under light workloads, finding good static topologies is important for two reasons:

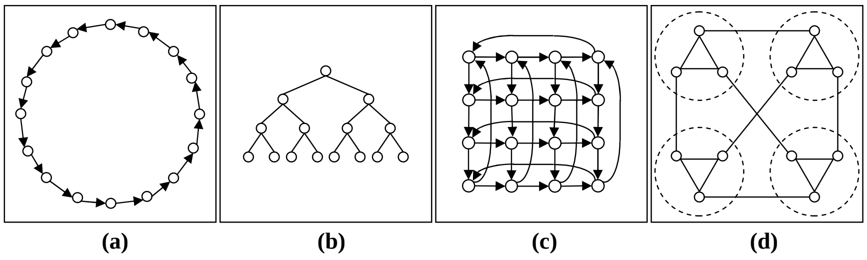


Figure 5.2: Indirect peer-to-peer routing topology graphs. Each node represents a Postmanet site. A directed edge indicates that data can only flow in the direction of the arrow. An undirected edge indicates that data can flow in either direction.

(1) a good static routing graph may form the basis of a good dynamic routing algorithm; and (2) a good static routing graph may provide a performance upper-bound for a uniformly heavy workload, which may present few exploitable optimization opportunities for any dynamic approach.

Figure 5.2 shows some ways that Postmanet nodes can forward data for each other. Suppose the number of Postmanet nodes is N. Using the ring topology of Figure 5.2(a), a Postmanet user handles one disk but the end-to-end latency can be long. Using the tree of Figure 5.2(b), a Postmanet user handles three disks and the end-to-end latency is $\log N$, but the higher-level nodes are likely to need to copy more data. The ring topology of Figure 5.2(a) can be extended to a 2-d mesh as shown in Figure 5.2(c), or more generally, to a k-dimensional mesh. In such a topology, a Postmanet user handles k disks and the worst case end-to-end latency is $k\sqrt[k]{N}$. Figure 5.2(d) illustrates a hierarchical cluster-based algorithm for constructing a routing topology: an "$(i+1)$th level cluster" is a complete graph of $j+1$ lower-level (ith level) clusters, each of which has j nodes inside. In Figure 5.2(d), for example, $i = 1$ and $j = 3$. At the next two levels up, j grows to $3 \times 4 = 12$ and $12 \times 13 = 156$, respectively. In this topology, the worst case end-to-end latency is $O(\log N)$, and the number of disks handled by each user is $O(\log\log N)$.

Other alternatives include the various interconnection topologies used by Distributed Hash Tables (DHTs) to enable object location in peer-to-peer systems. These include adaptations of hypercubes [54, 72], tori [52], and de Bruijn graphs [31]. The hypercube-based interconnections, such as Pastry and Tapestry, can be utilized in a Postmanet to probabilistically limit the end-to-end latency to $O(\log N)$ with each user handling $O(\log N)$ disks. Topologies based on de Bruijn graphs can also probabilistically limit the end-to-end latency to $O(\log N)$ while requiring only a constant number of disks handled by each user. These systems employ implicit routing wherein routing decisions are made locally without requiring elaborate knowledge of the global topology. We do however note that implicit routing may be of limited value in Postmanet, where the control and data traffic can be conveyed on different networks—the LLLB Internet could be used for dispersing topology information or topology repairs in case of failures, while bulk data is communicated over the HLHB channels.

Recall that the graph diameter corresponds to the maximum end-to-end latency, and the degree of a node corresponds to the number of disks it handles. For diameter D and maximum degree Δ, it can be shown that an upper bound (known as the *Moore bound*) on the number of nodes in the graph is: $(\Delta^{D+1} - 1)/(\Delta - 1)$. The Moore bound can not be achieved exactly, except for the trivial cases when Δ or D is 1. There is, however, a long history of attempts to explicitly construct large graphs with bounded diameter and maximum degree [22]. Many of these constructions use sophisticated group theoretic techniques. Randomized constructions with constant degree for each node and $O(log\ N)$ diameter are also well known. We, however, note that it is important to map these abstract graphs onto Postmanet participants in a geography-aware manner so that we do not incur unnecessary extra postal system delays and costs. Therefore, simpler and more intuitive

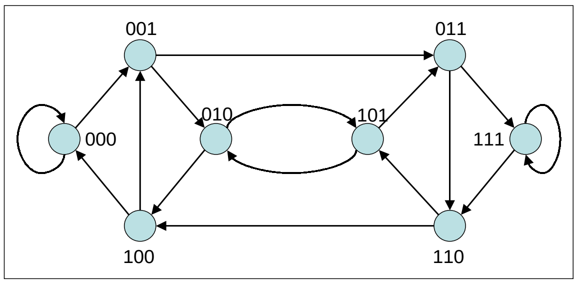


Figure 5.3: A 3-dimensional de Bruijn graph.

constructions, such as those illustrated in Figure 5.2, may eventually prove to be more usable in practice.

We examine two types of static routing graphs for use in the Postmanet: de Bruijn graphs [11] and random graphs. An r-dimensional de Bruijn graph consists of 2^r nodes. Each node is associated with a distinct r-bit binary string, and a node identified by the binary string $b_1 b_2 \cdots b_r$ has directed edges leading towards the nodes identified by $b_2 \cdots b_r 0$ and $b_2 \cdots b_r 1$. (Figure 5.3 illustrates an 8-node de Bruijn graph.) Each node has both an indegree and outdegree of two. To route from a node $u_1 u_2 u_3 \cdots u_r$ to a node $v_1 v_2 v_3 \cdots v_r$, one simply routes through the intermediate nodes $u_2 u_3 \cdots u_r v_1$, $u_3 \cdots u_r v_1 v_2$, $\ldots$, $u_r v_1 v_2 \cdots v_{r-1}$, thereby resulting in a system with diameter of $\log N$. When the number of nodes involved is not an exact power of 2, a static routing graph can still be obtained by starting with a larger de Bruijn graph, and “routing through” non-existent nodes. Also note, choosing k as the “base” would result in a degree-k graph.

Although random graphs can also achieve $O(log\ N)$ diameter with constant node degree, unlike de Bruijn graphs, the diameter bound is probabilistic. Furthermore, compared to de Bruijn graphs, random graphs tend to require a larger node degree constant to achieve a comparable diameter bound.

5.3.2 Disseminating Traffic and Routing Information

In traditional networks, implicit routing, wherein routing decisions are made locally without requiring elaborate knowledge of the global topology, can be very useful. In contrast, implicit routing may be of lesser importance in a Postmanet that has two “networks”—the LLLB Internet could be used for dispersing topology information or topology repairs, while bulk data traverses the HLHB channels.

In a similar vein, we can also use the LLLB channel to disseminate traffic information (in terms of who desires to send bulk data to whom). This makes the dynamic routing problem easier to solve. We may assume, for example, that traffic information is continuously being gathered at a centralized coordinator site over an LLLB channel. The coordinator uses the gathered information to compute the best dynamic routes, which the coordinator then disseminates to all the participating peer Postmanet sites, so by the time a postman arrives at a site to pick up outgoing P-disks, appropriate next-hop postal labels would have been generated at each site according to a global schedule and affixed to these outgoing P-disks. Furthermore, as much as 24 hours, for example, may elapse between successive postman visits, so the coordinator may have ample time computing the best dynamic routes. Multiple coordinators can be employed to improve reliability and performance.

5.3.3 Dynamic Routing

In Section 5.3.1, we have described a static routing strategy, where a message from a node s destined for a node t is always routed along the shortest path in the underlying de Bruijn graph. As pointed out in Section 5.2, this can be overly restrictive, especially when some nodes are lightly loaded. Consider the example in Figure 5.4. The figure shows a portion

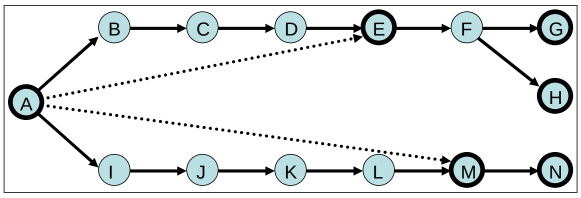


Figure 5.4: A dynamic routing example. The dark arrows are the edges in the underlying static routing graph. The dotted edges are the "short-cut" edges that A uses to directly forward messages to E and M.

of the underlying static graph, where out- and in-degree of each node is constrained to be at most two. (Not all graph edges are shown here.) Assume that at some stage, node A only has messages destined for nodes E, G, H, M and N. In this case, instead of using the edges in the underlying graph, A may use the "short-cut" edges $A \rightarrow E$ and $A \rightarrow M$, and in a single step, send the messages destined for E, G and H directly to E, and those destined for M and N directly to M.

A good "dynamic routing algorithm" for Postmanet would make decisions of this kind in an optimal manner. Specifically, it would be an "on-line" algorithm that for each postman visit at each site, determines the next-hop destinations for the out-going P-disks, and also selects the set of messages to put on those P-disks. In general, an "on-line" optimization problem needs to be solved to dynamically choose the short-cut edges along which P-disks are to be shipped, and to select the messages to put on those devices. The goal is to make progress toward delivering messages to their respective destinations, while respecting the degree constraints on the nodes per postman visit. As new messages are generated each day, the state of the system needs to be updated, which affects the choice of the short-cut edges on subsequent days. One way to measure incremental "progress" made by the system toward delivering a given message is to measure how close in the underlying static graph the message has reached to its eventual destination. A greedy optimization algorithm can then, at each step, try to choose the edges so as to quickly make as much global progress as possible.

In our proposed dynamic routing approach, an algorithm is run at the end of each step (or day) to determine the edges along which to ship P-disks on the next day. Our algorithm constructs a bipartite graph with vertex set $P \cup Q$, where each node of the system appears exactly once in both P and Q. Edge $p \rightarrow q$ is assigned a weight proportional to the progress that can be made by sending a P-disk from node p to q directly. The problem then reduces to choosing a set of edges so as to make as much total progress as possible. For this purpose, maximum-weight matching algorithms, or min-cost max-flow algorithms can be used. The algorithm is repeatedly invoked to find a set of matchings along which to ship P-disks. The weights on the edges can also take into account other factors such as message priorities, delivery deadlines and starvation. With a suitable choice of the progress metric, the dynamic routing algorithm would degenerate to the static routing algorithm under heavy message traffic. (Specific progress metrics will be discussed in a later section.) Thus, in the worst case, the performance of the dynamic algorithm would be no worse than that of the static algorithm, while under lighter load conditions, the dynamic algorithm would perform much better.

It is interesting to note that under this dynamic routing approach, the dynamically chosen routes are by no means obliged to follow any edges in the static underlying de Bruijn (or random) graph. The static graph's sole purpose is providing a means for the dynamic algorithms to gauge progress when greedily choosing next hops. In some sense, the static graph acts as a "traffic shaper," whose influence should be the strongest under extremely heavy workloads, which we conjecture would force the dynamically chosen routes to more closely conform to the shortest paths in the static graph.

Although the role of a static graph is only a traffic shaper, it is important for the static graph to have a node degree constraint that is identical to that of the dynamic routing graph, which should reflect the real-life limitation of how many P-disks a site handles. Had we chosen a static graph with a higher node degree and enforced a lower node degree only in the dynamic routing algorithm, the progress metric derived from the static graph could be overly optimistic, potentially resulting in too many messages being delivered to a site that cannot drain them quickly due to a low dynamic degree limit.

It is possible to model dynamic routing as a more precise optimization problem, and to try to achieve "theoretically-optimal" solutions. However, there seems to be little hope of finding such optimal solutions for two reasons. First, even the off-line version of our problem (where all of the requests are available at the beginning of the computation) appears to be NP-hard because of its relationship with the well-studied multi-commodity fixed-charge problems. Second, our real interest lies in devising an on-line algorithm that executes continuously and handles a multitude of events occurring in the system, and not all versions of our problem may be easily amenable to theoretically optimal solutions. Hence we use a greedy, heuristics-based approach.

5.3.4 Geographic Awareness

The postal system latencies for different source-destination pairs cannot be assumed to be uniform in general. For example, depending on the shipping option used, even within the continental United States, the delivery times can vary between 1 day and as many as 7 days. Therefore, Postmanet routing needs to be aware of the non-uniform postal system latencies to prevent routing inefficiencies.

We use two techniques to make Postmanet routing geography-aware. First, we embed the static routing graph onto the set of participating nodes in a geography-aware fashion such that the postal system latencies along the graph edges are not too large. In other words, our algorithm tries to map neighboring nodes in the graph to physical nodes that have small postal system latency between them. This is achieved by using a Dijkstra-style greedy algorithm that tries to ensure that the postal system latencies along the graph edges are not too large. Second, actual postal system latencies are taken into account when assigning weights to the edges in the bipartite graph used in the maximum-matching computation. We study the effects of both of these techniques in chapter 5.4.

5.3.5 Integrating P-Centers

We next consider how to integrate P-centers into our peer-to-peer routing infrastructure. P-centers, with their ability to provide two-hop connectivity between any pair of nodes, could be used to either service only some high-priority messages, or improve the latency of all messages by providing short-cuts in the routing infrastructure. The optimization problem, in either case, is to compute a set of source nodes and a possibly overlapping set of destination nodes for which a given P-center would serve as a hub on a given day in order to maximize the progress of the messages in the system. Each node is constrained to send to (or receive from) the P-centers at most one P-disk on any given day. A given P-center would not be statically bound to a fixed set of nodes, thereby allowing it to adapt to varying traffic conditions. Once again, theoretically optimal solutions for this problem are intractable even for the special case of augmenting the infrastructure with just one P-center, and we therefore resort to the following heuristic.

We determine the routing connectivity for one P-center at a time during each routing step. We begin by greedily picking a source node that would attain the greatest benefit from using the P-center to communicate its messages to at most d_{pc} destinations, where d_{pc} is the out-degree constraint on the P-center. We then pick the next source node based on a metric that takes into account both the amount of message traffic to some set of d_{pc} destinations and the amount of message traffic to only those destinations that are favored by the first selected node. We repeat this process, and at each step, we keep track of the most popular destinations corresponding to the current set of selected source nodes and pick the next source node based on this information. Once we have picked all the source nodes, the most popular d_{pc} destinations are selected as the target nodes to which the P-center will send a P-disk. Based on the final selection of the target nodes, each source node will then compute what messages it will send to the P-center node, including on the P-disk any message that would make faster progress through the P-center than through the peer-to-peer infrastructure.

5.4 Routing Simulation

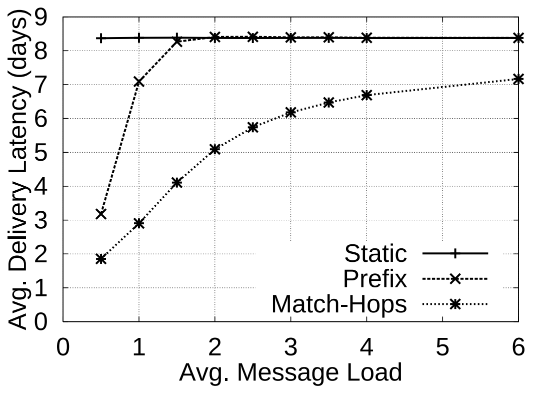
We have evaluated our proposed routing methodologies via simulation. We have developed an event-driven simulator to study the various routing strategies described in this chapter. Our simulator allows us to systematically evaluate the performance and scaling properties of the various algorithms under different workloads, study the effects of using different kinds of static graphs, examine ways of mapping abstract graphs to real-life Postmanet configurations, and evaluate the benefit of integrating P-centers into the routing infrastructure.

Although our current Digital StudyHall testbed is in India, we did not have access to good Indian postal data, so the study is performed based on U.S. postal data. As we discuss below, however, one would see that the assumptions here may very well apply to India. The nodes used by our simulator correspond to randomly chosen USPS zip codes, located at real-life geographic coordinates. The simulator uses a latency matrix, enumerating latencies between all pairs of nodes. We examine two types of latency matrices. In one type, all latencies are equal to one day. This “uniform latency matrix” corresponds to a fast delivery service (such as FedEx, or courier services in India). In another type, the latencies are set to be proportional to the geographical distances between nodes. At one day per 500 miles, with a maximum latency of eight days in the lower 48 states and a minimum of one day, and with the inclusion of nodes in Hawaii and Alaska, this second type of matrix represents a pessimistic assumption of the delivery service speed, a speed that is in fact worse than that experienced by DVDs delivered as first class USPS mails, and perhaps more similar to postal speeds seen in India. We shall refer to this as the “USPS latency matrix.” By choosing to use these two very different types of latency matrices, we hope to get some idea on the range of Postmanet latencies one might see in real-life.

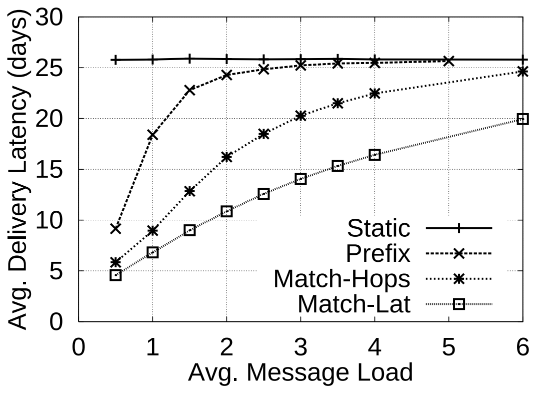
For the experiments in this section, we use a parameterized, random workload, where each node generates λ new unit-sized messages each day destined for λ distinct, randomly-selected other nodes. The parameter λ is referred to as the “Average Message Load” of the workload, or simply as the “load.” All workloads in our study contain 60 days worth of message traffic. To account for our conjecture that people either do not communicate or communicate with more than an average number of other parties, we have introduced “burstiness” into the workload. Hence a node would generate a lot of messages on some days and very little during others, resulting in “bursty” traffic but with the average load given by the parameter λ.

5.4.1 Routing Algorithms

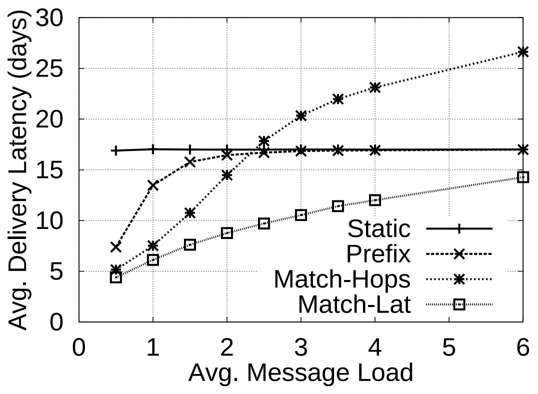
We now compare static and dynamic routing algorithms, and also study the impact of using different progress metrics for the dynamic algorithms. In the *Static* algorithm, each node only sends P-disks to its neighbors in the underlying static graph each day. The *Prefix* algorithm is a dynamic algorithm that chooses “short-cut” edges that correspond to multiple hops in the underlying static graph, as discussed in Section 5.3.3 and illustrated by Figure 5.4. The chosen short-cut edges are, however, constrained so that no message ever overshoots its destination. For example, in the topology of Figure 5.4, suppose A has 1 message each for E and M, and 1,000 messages each for G and N. Although edges $A \rightarrow G$ and $A \rightarrow N$ make greater incremental progress in terms of delivering messages, the *Prefix* algorithm is constrained not to overshoot E and M, thus choosing edges $A \rightarrow E$ and $A \rightarrow M$. The *Match-Hops* algorithm is a dynamic algorithm that is not hobbled by the above constraint, but instead uses a maximum-weight matching technique to maximize the sum progress of all the messages through the network. (See Section 5.3.3.) The progress metric associated with transmitting a message over an edge is simply how much closer the message is to its eventual destination in terms of number of hops in the static graph. The *Match-Lat* algorithm uses a different progress metric that takes into account the postal system latencies (and not just hop-counts) in determining how much closer the message is to its final destination in the static graph. Our implementation of these algorithms uses Goldberg’s Network Optimization Library [20].



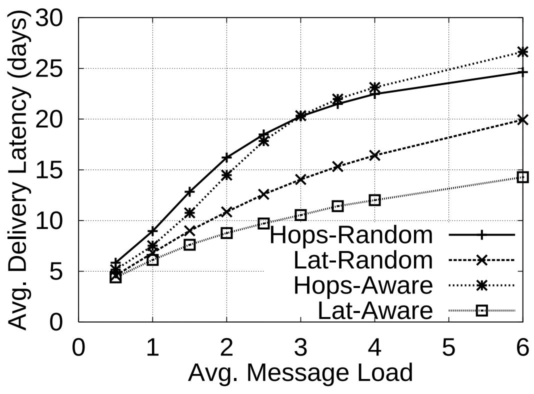
(a)



(b)



(c)



(d)

Figure 5.5: Comparison of different routing algorithms and geography-awareness techniques. The runs in (a) use the uniform latency-matrix, whereas those in (b) and (c) use the USPS latency-matrix. A random mapping of physical nodes to de Bruijn graph nodes is used in (b), while (c) uses a geography-aware mapping. (d) plots some curves from (b) and (c) on the same scale for comparison.

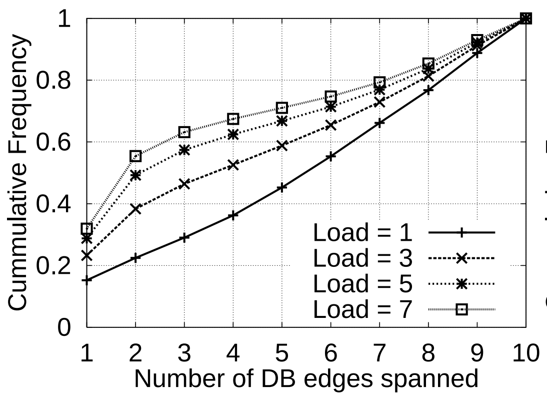
Figure 5.5(a) shows the performance of three routing algorithms for a network comprising of 1,024 nodes. A de Bruijn graph of degree two (referred to as “DB-2”) is used as the underlying static graph, and the uniform latency-matrix is used to specify inter-node latencies[1]. We vary the “load,” which is the average number of messages generated at each node on each day in the workload, and measure the average message latency (in days). (The latency in the figure is greater than one even when load is no greater than one because of the workload burstiness: the instantaneous load tends to be higher than average when a node communicates.) The following observations can be made from this graph. (1) *Static* uses only the edges in the static graph, and yields an average latency that is precisely the average distance between a pair of nodes in the underlying static graph, a value that does not vary with the load. (2) The two dynamic algorithms perform much better than *Static* when the network is lightly loaded. As the load increases, their performance gracefully degrades and approaches that of *Static*. (3) *Prefix* degenerates to *Static* as soon as the average load approaches the number of P-disks that each node handles (two in this case), whereas *Match-Hops* out-performs *Static* for a much wider range of load values.

Figures 5.5(b) and 5.5(c) present results from similar executions, except that here the USPS latency-matrix is used instead of the uniform one. In Figure 5.5(b), a *geography-unaware,* random embedding is used to assign the physical nodes to the de Bruijn graph nodes, whereas in Figure 5.5(c) a *geography-aware* mapping generated via a Dijkstra-style greedy algorithm is used. Figure 5.5(d) plots the *Match-Hops* and *Match-Lat* curves from Figures 5.5(b) and 5.5(c) on the same scale to aid us in the task of comparing the different schemes. We begin by observing that *Static* performs much better in Fig-

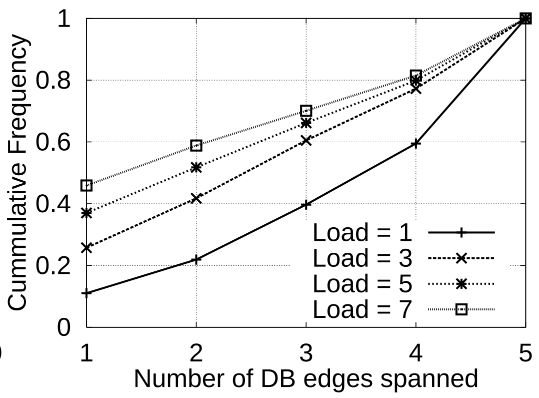
[1]For the uniform latency-matrix, *Match-Lat* has the same behavior as *Match-Hops,* and is therefore omitted from the figure.

ure 5.5(c) than in Figure 5.5(b), which is evidence that our geography-aware greedy algorithm produces a mapping that has significantly smaller postal latencies along the graph edges than a random mapping. We also observe that *Match-Lat* significantly outperforms *Match-Hops*, mainly because the latter algorithm uses a progress metric that is oblivious to the non-uniform nature of the postal latencies. This, combined with its greedy nature, makes *Match-Hops*'s performance even worse than that of *Prefix*, which benefits from its conservative approach of not overshooting its destinations and simply degrades to using the static graph edges in the worst case. Finally, we observe from Figure (d) that the *geography-aware* version of *Match-Lat* performs the best overall.

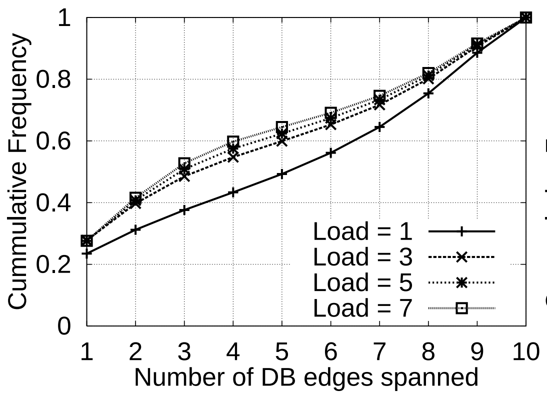
In Section 5.3.3, we conjectured that the use of a static underlying graph acts as a "traffic shaper" for the dynamic algorithms, especially under heavy load conditions. Figure 5.6 presents evidence to support this. Consider Figure 5.6(a) first. It describes executions of the geography-aware *Match-Lat* algorithm on four workloads of different average loads. In each execution, we count the number of times the algorithm picks a short-cut edge that spans k de Bruijn edges, and the curve is a *cumulative frequency distribution* of these counts. In other words, a data point (x,y) on a curve signifies that y% of the short-cut edges selected by the algorithm span x or fewer de Bruijn edges. For example, when 100% of the chosen edges span only one de Bruijn edge, we effectively have a traffic that entirely flows along the de Bruijn graph. Looking at the four curves, we observe that as load increases, a higher fraction of the chosen edges span only a small number of de Bruijn edges; that is, under high loads, the dynamic traffic more closely conforms to the underlying de Bruijn graph. Figures 5.6(a)-(d) show results from different de Bruijn graphs and different latency matrices, and they all support the same conclusion.



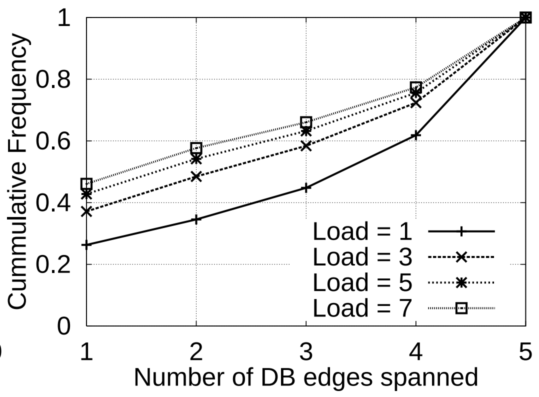
(a)



(b)



(c)



(d)

Figure 5.6: Analyzing the traffic generated by the geography-aware Match-Lat algorithm (a) on a DB-2 graph with the uniform latency matrix, (b) on a DB-4 graph with the uniform latency matrix, (c) on a DB-2 graph with the USPS latency matrix, (d) on a DB-4 graph with the USPS latency matrix.

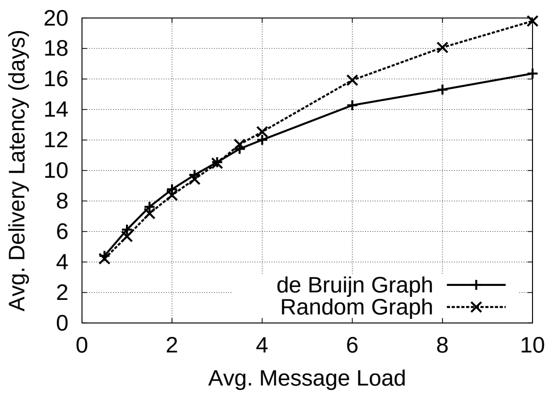


Figure 5.7: Comparing a degree 2 de Bruijn graph with a degree 2 random graph as the underlying static graph for the Match-Lat algorithm. We use the USPS latency matrix for this experiment.

5.4.2 Comparing de Bruijn and Random Graphs

Figure 5.7 shows a comparison between using a de Bruijn graph or a random graph as the underlying static graph. The curves show the performance of the geography-aware *Match-Lat* algorithm where each of the 1,024 nodes has degree 2. As the graph shows, de Bruijn graph-based execution performs better than that based on a random graph under high message loads. Here we omit results that show that the difference between the two graphs is less pronounced when each node has degree 4 or more, or when the uniform latency matrix is used. The degree 2 case shown in the figure is realistic enough to make our use of the de Bruijn graph worth-while.

5.4.3 Integrating P-Centers

We now present some preliminary results of the impact of integrating data distribution centers (or P-centers) into the peer-to-peer infrastructure. Figure 5.8 shows the performance of a dynamic routing algorithm (*Match-Lat*) on a 1,024-node network that uses a de Bruijn graph of degree two as the static underlying graph. The workload under consideration generates on average five messages per node each day. We vary the number

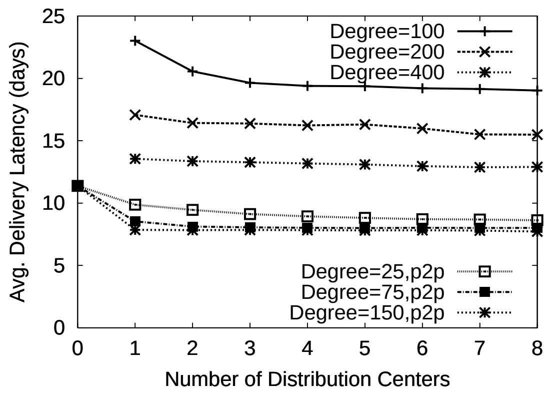


Figure 5.8: Impact of distribution centers on routing performance.

and the degree capacity of the P-centers and evaluate the routing performance under the two following settings: (1) the P-centers serve all of the message load in the system (with the peer-to-peer infrastructure remaining unused), and (2) the P-centers share the routing load with the peer-to-peer infrastructure. For instance, the curve with the legend “degree=150, p2p” corresponds to augmenting peer-to-peer routing with P-centers that can handle 150 incoming P-disks and generate 150 outgoing P-disks every day, while the top three curves in the figure correspond to using just P-centers for routing.

In these preliminary results, we see that a modest start of augmenting the peer-to-peer infrastructure with just one P-center causes a modest improvement on performance, but the marginal benefit of either increasing the number of P-centers or their degree capacity appears small. In particular, as we increase the number of P-centers, although we are making more routing resources available, our current heuristics may be simultaneously increasing extra hops among the P-centers themselves, thus partially negating the advantages of the extra P-centers. We also see that a single P-center with a large degree capacity appears to perform better than having more P-centers with a smaller degree capacity. We also see that the peer-to-peer infrastructure, with no additional P-centers, appears to outperform many stand-alone P-centers. This could potentially be explained by the fact that in a large geographic area, an end node is likely to be still relatively far away from the nearest of the modest number of P-centers, while the end nodes may likely have shorter paths among themselves, especially when they take the de Bruijn "short cuts." This effect may not be the case within a smaller geographic area served by a single content production hub in the Digital StudyHall today. We caution, however, that the heuristics that we are using currently for incorporating the P-centers are directly derived from the peer-to-peer algorithm, and thus may not be fully taking advantage of the P-center resources; and better approaches and better results may be possible than what is implied in this figure.

Chapter 6

A Message Passing API

Our experiences with the Digital StudyHall system (Chapter 3), built on top of the PHTTP substrate (Chapter 4), have exposed a number of issues that we need to work on. In the previous chapter (Chapter 5), we have studied the issue of scalable routing strategies, strategies that are necessary if we were to expand the Digital StudyHall "network." In this chapter, we return to the end points, and examine a number of issues that have not been adequately addressed in the PHTTP layer, issues that are unique to the Postmanet environment. These issues include the handling of bursty arrival of bulk data, the intelligent exploitation of two networks in parallel, support for undo operations, and P-disk physical organization for optimal performance. We shall see that even just from a performance point of view, the answer of "leaving it to the applications" is far from adequate, and we need system-level solutions that support and coordinate needs from multiple applications.

For this part of the study, we have done our work on an alternative message passing API, on which we have developed a number of experimental applications other than the Digital StudyHall. Several practical reasons underlie this decision. The Digital StudyHall and the PHTTP systems provide the underlying support infrastructure of a larger rural distance learning project in India today. As such, it is a production system whose maintenance and development need to be carefully managed, so we do not disrupt daily education work. As a result, it is not the best platform for ad hoc experimentation. The study discussed in this chapter has been done in parallel to the work done on the Digital StudyHall, so it has not always been possible to "graft" the ideas discussed here onto a mature version of the PHTTP substrate during the development of both. Not all applications are browser-based, and the message passing API discussed in this chapter potentially allows a greater degree of freedom. Examination of other applications beyond the Digital StudyHall application, albeit much simpler ones, is important, given our goal of engineering *general* systems support for the Postmanet. Our intention is to investigate a number of experimental approaches to the issues mentioned earlier, and only after we have understood the implications of our approaches in this alternative message passing API, will we consider grafting these ideas back onto the production PHTTP system.

6.1 Postmanet Characteristics and Implications

The Postmanet has several unique characteristics, which require treatment different from that in conventional networks.

• *Bursty arrival of large amounts of data.* A single Postmanet sender could have spent many hours writing to a P-disk, and data from multiple sources can arrive at a receiver per postman visit. Gigabytes or even terabytes of data could be involved. A (human or application) receiver would naturally desire to gain access to the newly arriving data as quickly as possible. A naive approach of forcing the receiver to wait until the system completes copying from incoming P-disks to local storage could add substantial delay. Instead, it is important to allow receiver applications quick access to summary or metadata information so that they can make flexible decisions before a large amount of data needs to be copied. This is an issue that does not arise in conventional networks that allow gradual and continuous data arrival.

• *Two networks.* The villages where DSH is deployed today do not yet have any other modes of connectivity than the Postmanet. In the future, perhaps elsewhere, the aid of an LLLB Internet connection should make the Postmanet operate better. In addition to using the LLLB channel to carry small control messages, the sender system may choose between the LLLB Internet and an HLHB P-disk based on factors such as the amount of data to be sent and the desired arrival time. One may even choose to use both channels in parallel. For example, a Postmanet application may prepare multiple versions of an object (at different resolutions) for simultaneous transmission on the LLLB and HLHB channels. The LLLB link is particular useful in dealing with the datagram limitations of the Postmanet: the postal system represents a classic analogy of a datagram service—individual P-disks may be damaged, lost, delayed, or delivered out of order. In PHTTP today, these complications are solely dealt with by the control messages (such as acknowledgements) carried by P-disks, and the use of LLLB links for this purpose would have improved the system.

• *Delayed action.* Conventional networks typically do not support, for example, an “unsend” operation, that allows a user to change his mind after a “send” operation is executed, because there is typically little time before actions are effected. In the Postmanet, however, there is ample opportunity for mind-changing: before the postman picks up the outgoing P-disk at the sender, as the P-disk is in transit in a P-center or in a peer’s P-router, or even after the P-disk arrives at the destination but before the data is consumed by the receiver application. Even in absence of a mind-changing sender, a receiver application may discover that some of the newly arriving data is no longer needed due to application-specific reasons. In any case, the LLLB channel can be used to "shoot down" a message in any stage of transmission between the sender and the receiver end-points.

• *P-disk communication media.* Large-capacity P-disks play the role of wires. A P-disk may hold many messages, which require the data to be organized in a more structured fashion than that typically employed on a wire. A natural question is what type of structure we should use: for example, a database, a file system, or some other customized data structure? The physical organization of storage management is also a relevant issue. For example, a log-structured approach [53] may allow small message "sends" and certain types of copying to execute efficiently. PHTTP simply uses the native CD/DVD file system to store its data and metadata. As we shall see, its performance can be far from optimal.

6.2 API Overview

The most important means of addressing the unique Postmanet characteristics discussed above is well-defined APIs. Properly defined APIs should (1) abstract away unpleasant details (such as the datagram limitations of the postal services), (2) expose new capabilities (such as ways of using two networks), and (3) allow applications to circumvent performance difficulties (such as the problems associated with bursty arrival of large amounts of data). We give an overview of the interfaces, before we later explain how they address the Postmanet-specific characteristics.

There are three key sets of interfaces: (1) an interface that allows applications to manipulate data on P-disks (Figure 6.1), (2) an interface that controls sending and receiving of data (Figure 6.2), and (3) an internal interface used by P-routers that communicate with each other (but not visible to applications) (Figure 6.3).

1. **`Entry.getName()`**
   name of this `Entry`*.*
2. **`Entry.getFullName()`**
   full path of this `Entry`*.*
3. **`Entry.create(name, type)`**
   create subentry. `type`: `File` *or* `Dir`*.*
4. **`Entry.delete(name)`**
   delete a subentry.
5. **`Entry.deleteAll(name)`**
   delete a subentry recursively.
6. **`Entry.get(name)`**
   returns the specified subentry.
7. **`Entry.list()`**
   list names of subentries of this `Entry`*.*
8. **`Entry.listEntries()`**
   returns list of subentries in this `Entry`*.*
9. **`Entry.search(filter)`**
   list subentries that match `filter`*.*
10. **`Entry.isFile()`**
    test whether this `Entry` *is a* `file`*.*
11. **`Entry.isDirectory()`**
    test whether this `Entry` *is a* `Dir`*.*
12. **`Entry.size()`**
    returns size of this `Entry`*.*
13. **`Entry.getFD()`**
    gets the "file descriptor" for this `Entry`*.*

(a) **`Entry`** interface (partial).

1. **`FD.lseek(offset, whence)`**
   set this pointer to specified `offset`*.*
2. **`FD.getOffset()`**
   returns current offset.
3. **`FD.length()`**
   returns size of this `File`*.*
4. **`FD.sync()`**
   flush changes to disk.
5. **`FD.read(bytes)`**
   returns next `bytes` *of data.*
6. **`FD.read(count,prefetch)`**
   similar. `prefetch`*: this is low priority.*
7. **`FD.write(data)`**
   write given data to this `File`*.*
8. **`FD.write(srcFD, offset, bytes)`**
   write `bytes` *starting from given source* `srcFD`*'s given* `offset`*.*

(b) **FD** interface.

1. **`Entry.getAttributes()`**
   returns all attributes of an this `Entry`*.*
2. **`Entry.getAttributes(nameSet)`**
   returns only specified attributes.
3. **`Entry.getAttribute(key)`**
   returns specified attribute.
4. **`Entry.modifyAttribute(key,value)`**
   modify/add specified attribute.

(c) **`Entry`** attribute interface.

Figure 6.1: Local storage interfaces (available to applications).

1. **`EP.getZip()`**
2. **`EP.setZip(pzip)`**
3. **`EP.getMailbox()`**
4. **`EP.setMailbox(mailbox)`**
5. **`EP.getIP()`**
6. **`EP.setIP(ip)`**
7. **`EP.getAddress()`**
   get postal address for this `EndPoint`*.*

(a) **`EndPoint`**interface.

1. **`Msg.newMessage()`**
   start a new message.
2. **`Msg.newMessage(path)`**
   start a new message with an `Entry` *of a given* `path`.
3. **`Msg.newMessage(entry)`**
   used by a receiver to treat a received `entry` *as a message.*
4. **`Msg.add(sourcePath, destPath)`**
   add `sourcePath` *as* `destPath`*.*
5. **`Msg.add(entry,destPath)`**
   add `entry` *as* `destpath`*.*
6. **`Msg.setRecipients(endPointList)`**
   set message destinations.
7. **`Msg.setReturnAddress(endPoint)`**
   set return address for `Ack`*s etc.*
8. **`Msg.setTracking(level)`**
   set message tracking to given `level`*.*
9. **`Msg.setDeliveryDeadline(date)`**
   allows application to set a hint.
10. **`Msg.setReplicaID(id)`**
    for identifying application-level replicas.
11. **`Msg.setInternetDelivery()`**
    hint: delivered over the LLB Internet.
12. **`Msg.setResolution(level)`**
    set resolution `level` *of the message.*

(b) **`Message`** interface (partial).

1. **`Pnet.send(msg, callback)`**
   asynchronous send. `callback` *is invoked when* `msg` *has been copied to local spool (or an error occurs). returns a* `MessageID`*.*
2. **Pnet.send(msgID, `msg, callback)`**
   similar. sends with specific msgID *. Useful for sending "same messages" at multiple resolutions.*

(c) **`Postmanet`** send calls.

1. **`Pnet.delete(msgID)`**
   delete a message.
2. **`Pnet.delete(msgID, filter)`**
   delete parts of message matching `filter`*.*

(d) **`Postmanet`** delete calls.

1. **`Pnet.getRootEntry()`**
   returns root `Entry`*.*
2. **`Pnet.setCallback(mailbox, callback, filter)`**
   set `callback` *for a given* `mailbox`*.* `callback` *is invoked everytime a message matching filter is received.*
3. **`Pnet.removeCallback(mailbox, callback)`**
   remove a previously set callback.
4. **`Pnet.getCallbacks(mailbox)`**
   lists all `callback`*s in effect for this mailbox.*
5. **`Pnet.getNext(mailbox, flag)`**
   returns next new message in `mailbox`*.*

(e) **`Postmanet`**: receiver setup.

1. **`Msg.getSender()`**
   returns `EndPoint` *of sender.*
2. **`Msg.dateSent()`**
   returns date when message was sent.
3. **`Msg.getType()`**
   query whether message is an `Ack`*,* `TrackingUpdate`*,* `FailureNotice` *etc.*
4. **`Msg.getID()`**
   returns a `MessageID`*.*
5. **`Msg.discard()`**
   delete this message.

(f) **`Message`**: calls available to a receiver.

Figure 6.2: Communication interfaces (available to applications).

1. **`Peer.msgOk(messageID)`** *acknowledges the receipt of a message.* 2. **`Peer.msgError(messageID, cause)`** *error receiving a message due to* `cause`. 3. **`Peer.PdiskOk(PdiskID)`** *acknowledges the receipt of a P-disk.* 4. **`Peer.PdiskError(PdiskID, cause)`** *error receiving a P-disk due to* `cause`. 5. **`Peer.stashedCopy(messageID)`** *reports a copy of a message is stashed.* 6. **`Peer.discardedCopy(messageID)`** *reports a previously stashed copy is discarded.*	7. **`Peer.resend(messageID)`** *requests peer to resend a message. returns error if copy is no longer available.* 8. **`Peer.receiveEntry(Entry)`** *receives an* `Entry` *over the Internet.* 9. **`Peer.delete(messageID, filter)`** *delete parts of message matching* `filter`. 10. **`Peer.trackingUpdate(messageID, trackingInfo)`** *reports tracking information.* 11. **`Peer.expectReplica(messageID, replicaID, expiryDate)`** *informs that a replica is on the way.*

Figure 6.3: Peer interfaces (hidden from applications).

An `Entry` is a basic P-disk-resident object that roughly corresponds to a Unix file or a directory (6.1(a)). Unlike conventional files, however, additional Postmanet-specific semantics and operations are built on top of `Entries`. `FD`s, similar (but not identical) to file descriptors, allow data to be read from/written to P-disks (6.1(b)). Beyond the file system-like operations, an `Entry` also has associated "attributes," or `(Key, Value)` pairs, which are used by both the system and applications (6.1(c)). (As examples, the intended recipient identity would be a system attribute of the outgoing data; and the URL of a web-based publication would be an application-specific attribute.)

A `Mailbox` is a directory `Entry` under which an application finds incoming data. To send data, one needs to specify a destination `EndPoint` (6.2(a)), which contains a `Pzip` that identifies the receiver machine, a `Mailbox`, and optionally, the IP address of the destination machine. Such addressing information can be provided by a separate lookup service analogous to the DNS service of today. Lookups can leverage the LLLB Internet. We are using a simple local file in the current prototype.

`Messages` are `Entries`. A sender manufactures `Messages` using the interface of 6.2(b). These calls essentially allow one to set a variety of attributes, which specify various delivery options. Of all these calls, only `setRecipients()` is necessary. Because `Messages` are `Entries`, all supported calls of `Entries` (6.1(a)) are also available for `Messages`. Once a message is created, it can be sent using the calls of 6.2(c), or “unsent” using those of 6.2(d) (if one were to change his mind). Postmanet provides “reliable” messaging but it currently does not guarantee in-order delivery. It allows one-to-many communication.

To receive messages, an application sets `Callbacks` on its `Mailbox` (6.2(e)). When a `Callback` is invoked, the `Entry` that resulted in its invocation is passed as an argument to the callback function. Note that the callback function would need to explicitly perform read operations using the interfaces given in Figures 6.1 and 6.2(f). The data being read, however, may or may not have been moved from an incoming P-disk to other local storage at the receiver by the system. This allows both good performance, for applications that desire to control their own data movement from a P-disk into application-specific local store, and convenience, for applications that do not want to be bothered with such low-level details.

The peer interfaces shown in Figure 6.3 allow peer P-routers to communicate with each other, mainly over the LLLB Internet channel. These interfaces are not visible to applications. They allow P-routers to manage control information such as acknowledgements, failure notifications, retransmission requests, message shoot-downs, replica management, and message tracking. Small data messages are also transmitted over the LLLB Internet using this interface.

6.3 Managing Postmanet-Specific Characteristics

We now discuss how the unique characteristics of the Postmanet (Section 6.1) are managed by (and behind) the interfaces given above (Section 6.2). These characteristics interact in interesting ways: the problems caused by some of these characteristics can be addressed by opportunities offered by other characteristics. For example, the datagram limitations and poor latencies can be improved if we judiciously exploit the availability of two networks and the excess capacity on P-disks.

• *P-disk organization.* The "messages" are organized in a hierarchical file system, with additional attributes and supported operations added to the file system-like objects. This arrangement makes the system easy to use for applications, many of which would find a file system-like interface natural. A sending application can prepare the outgoing data in a format that its receiving counterpart can readily integrate into its own persistent data structures. Minimum packing, unpacking, or conversion should be necessary. What we are seeing is a form of blurring the boundary between storage and networks. (The issues being explored here, however, as we explain in chapter 7, are quite different from those seen in distributed storage and file systems.)

• *Two networks.* The LLLB Internet is made available for use to both the system and applications. The peer interfaces (Figure 6.3) allow peer P-routers to exchange various types of small control information. The `Message` interface allows applications to provide explicit or implicit hints (including delivery deadlines, and whether a message is a low resolution version of a bigger P-disk-resident message) on whether and when to use the LLLB Internet channel. (See calls 9, 11, 12 of Figure 6.2(b).)

• *Bursty arrival of large amounts of data.* We have several potentially conflicting goals: (1) copying all the data out of an incoming P-disk to receiver local storage as quickly as possible; (2) allowing applications to make progress without having to wait for extensive copying to complete; and (3) minimizing application interference with each other, which could result from competing data copying activities.

The callback interface (Figure 6.2(c)) allows applications to read a minimum amount of summary information to get started, and then to read data strictly on a need-driven basis. Behind the callback interface, a key P-router component is a generic system-level *background copier* that copies data from an incoming P-disk to local storage. If an application chooses to discard some incoming data (for any application-specific reasons) before it is reached by the background copier, this data does not need to be copied at all. As the background copier proceeds, the system ensures that the `Entry` passed to the application callback function points to the correct storage location, which could be either the P-disk or the local storage. The background copier may be able to aggressively exploit sequentiality of the underlying storage organization. The background copier, however, needs to exercise care not to compete with applications for P-disk bandwidth. The background copier also provides a means for applications to avoid interfering with each other: a "well-behaved" application should always read only what is necessary and leave the rest to the background copier. (This is based on the assumption that the receiver applications fed by the same P-router are willing to be cooperative, in terms of performance.)

• *Datagram limitations.* The handling of damaged, lost, delayed, or system-replicated Postmanet messages is not visible to the application-visible interfaces of Figures 6.1 and 6.2. The peer P-router interface of Figure 6.3 allows the system to quickly deal with these anomalies using the LLLB Internet. Furthermore, as multiple P-disks are sent between a sender-receiver pair on successive days, the system may liberally replicate outgoing data of earlier days on outgoing P-disks sent on later days. In cases where a single P-disk is delayed or lost due to accidents in the postal system or uncooperative peers who were supposed to forward it but did not, the replicated data on a subsequently arriving P-disk is just a day away, so we can avoid unnecessary long end-to-end retransmission delays. This is another example of the consistent Postmanet theme of liberally "wasting" plentiful resources (storage capacity) to optimize for more difficult metrics (lower latency or better reliability).

• *Delayed action.* A message can be canceled at any point after it is sent and before it is consumed at a receiver. Such a cancel message may need to be buffered at a destination. Shoot downs can be useful for functionality or performance reasons. Applications use the interface in Figure 6.2(d) to initiate a shoot down, which is handled locally if the message has not left the sender, or generates one or more peer shoot down messages (call 9 of Figure 6.3) if the P-disk containing the message has departed. Shoot downs can also be initiated at the system level without application initiation. For example, extra system-level replicas, such as those described in the last paragraph, should be shot down, when it becomes apparent that the outstanding replicas are no longer needed.

• *Security.* Security concerns can be largely addressed using existing mechanisms. Messages are protected against tampering and spoofing using the same techniques as used by PHTTP. For authenticating "shoot down" of in-transit messages, the node requesting the shoot down must present a "capability" for it signed by either the sender or recipient of the messages. The capabilities are exchanged via the LLLB.

In summary, the Postmanet has a number of characteristics not seen in conventional networks. We believe that careful interface design is an important way of addressing these issues. We do not, however, claim that we have arrived at the ideal interfaces, and we are continuing to evolve and refine these interfaces.

6.4 Implementation

We have implemented a prototype P-router in Java. All communication between applications and a P-router, and between P-routers is done via Java's Remote Method Invocation (RMI). The implementation contains three main modules: the sender part, which handles send and delete requests, the receiver part, which handles integration of data on incoming P-disks (or over the Internet) with the rest of the system, and a local store, which provides the `Entry` abstraction. The attributes of entries, and the message tracking and management information used by the above modules, is stored by an LDAP [46] server, running over a BDB [60] back end, accessed via JNDI. On the DVD P-disks, data is written as an ISO file system with attributes stored in separate files. The ISO images are staged on a local disk but partial images can be incrementally appended to DVD+RW discs that we use in our prototype. Using an ISO file system on the DVD allows us to retain usefulness of the data even outside of Postmanet system as no special software is needed to access data on it – which also helps during system development and debugging. The entire implementation is based on JDK v1.4.2_04 and is about 10,200 lines of java code.

6.5 Sample Applications

We briefly describe aspects of three simple additional Postmanet applications that we have developed and we highlight the key ways an application can leverage the functionality provided by Postmanet. The first one is PwebCache, a web proxy that receives subscribed data from a Postmanet-aware web publisher. The publisher creates data in an entry hierarchy to correspond to the on-disk structure used by the cache. The URL of the page, and cache validation and control headers of the HTTP protocol, are stored as attributes of entries and the `deliveryDeadline()` of the message is set to the cache expiration date of the data, if any. When a P-disk arrives, the receiver cache program only needs to record the URL attributes of the top level entries to be able to start servicing client requests immediately, while leaving the general system background copier with the task of moving data out of the P-disk.

The second application is Pnapster, a Napster-like application. The file lookup and request issuing parts are performed over the Internet and are no different from existing Napster-like applications. Multiple peers who have copies of the requested content may receive requests, so the requester may enjoy the quickest reply. A peer request receiver sets the `replicaID` to a "request ID," which allows Postmanet to manage (and shoot down, when necessary) these application-level replicas. When a "preview" request is received, a peer that has the desired data generates small (low-resolution) versions of the bulk data, invokes `setResolution()`, and `setInternetDelivery()` to hint it be sent over the Internet. A data requester may invoke `delete()` at any time to cancel a request.

The third application is Pemail, an email application. It uses `setTracking()` to acquire delivery status of outgoing messages. Pemail also generates small previews of bulk messages so the previews can be delivered over the Internet. A Pemail receiver may issue `delete()` calls with specific `filter` arguments to delete parts of messages before they are copied out of an incoming P-disk by the background copier.

DVD Writer	NEC ND2500A, 4× DVD-RW/+RW, 8× DVD-R/+R
DVD Media	Memorex 4× DVD+RW, 4.7 GB
OS	Linux 2.4.22 (Fedora Core 1)
Java	JDK 1.4.2_04
CPU	Pentium 3 800 MHz
Memory	128 MB
HDD	Maxtor 40 GB

Table 6.1: P-router machine characteristics.

6.6 Measuring the P-Router Prototype

We set up an old desktop machine (see Table 6.1) to function as a P-router. Performance-wise, perhaps the most interesting aspects are about how we handle the bursty arrival of a large amount of data (as discussed in Section 6.3), and we mainly focus on these aspects in this section.

We experiment with the three applications that we have described in Section 6.5. The sending applications create an outgoing DVD P-disk that contains the following data. A Pemail mailbox contains 100 messages, varying in size between 10-20 MB. The messages contain high-resolution images, home videos, and movie trailers. The PwebCache mailbox contains three messages. (Recall a message or an entry can be of a directory type that includes more sub-entries.) One message contains 85 MB of CNN.com news data; one contains 9 MB of data from news.yahoo.com; and one contains a travelogue and a photo gallery that totals 139 MB. There are eight Pnapster mailboxes, which contain a total of 48 messages, including mp3 files (which average 4 MB each) and one 500 MB avi movie. In all, the P-disk contains 152 messages, for a total size of 2.35 GB.

• *Basic operations.* Our P-router appends bulk ISO image data to DVD+RW P-disks at 4.7 MB/*s*, and reads bulk data from them at 3.17 MB/*s*. Sending a small entry, which is

Num. Mailboxes	8	10	10	10
Num. Messages	20	60	100	152
Data Size (MB)	80	366	966	2358
Case 1	34 *s*	172 *s*	458 *s*	1111 *s*
Case 2	134 *ms*	205 *ms*	253 *ms*	321 *ms*
Case 3	101 *ms*	103 *ms*	103 *ms*	103 *ms*

Table 6.2: Comparing startup times.

	news.yahoo.com	www.cnn.com
Naive	14.5 *s*	1047.9 *s*
Intelligent	6.6 *s*	256.3 *s*

Table 6.3: Exploiting knowledge of physical storage organization.

only written to the local staging disk, takes about 3 *ms* on average, while reading a small entry from a DVD+RW P-disk costs about 40 *ms*.

• *Quick startup.* When a P-disk arrives, it is important that the receiver applications (which are all interactive, in the case of our three example applications) can quickly access summary information so they can make their application-specific decisions about what to do with the incoming data, without being forced to wait for time-consuming mandatory system-level data copying to complete. Table 6.2 compares three cases. In "Case 1," the applications are given access to the data only after a P-router copier copies all the data from the incoming P-disk to a local disk. In "Case 2," the P-router iterates through all the entries on the P-disk, passing each entry to a null application callback function. In "Case 3," the P-router only iterates through all the mailboxes, passing only the per-mailbox summary information to a null application callback function. This experiment shows the importance of structuring the P-router and its applications in a way that can avoid mandatory copying or scanning of large-capacity P-disks upon their arrival.

	Pemail BW (MB/s)	Copier BW (MB/s)	DVD BW (MB/s)
Pemail alone	2.97		2.97
Pemail & dumb copier	0.17	0.17	0.34
Pemail & smart copier	2.73	2.34	2.66

Table 6.4: Potential impact of copier interference.

• *Exploiting knowledge of physical storage organization by the background copier.* Although it is important to allow applications to flexibly read from an incoming P-disk, a generic system-level P-disk copier may be able to function more efficiently by (1) exploiting knowledge of the physical storage organization (such as data locality) that applications are either unaware of or are unwilling to exploit due to complexity; and/or (2) performing more efficient scheduling across multiple applications. As applications dedicate the data movement tasks to such an efficient system-level background copier when possible, we may be able to drain data from incoming P-disks more quickly. In our prototype, the system background copier is able to exploit its knowledge of the ISO file system format, which clusters metadata in such a way that causes a naive recursive copier to suffer significant performance penalty. Table 6.3 shows an experiment of draining the PwebCache data from an incoming P-disk: the intelligent system-level P-router copier performs much more efficiently than a naive application-level recursive copier.

• *Cooperation between applications and the system-level copier.* We have argued above that both application-driven reads and system-level copiers are useful for efficiently draining incoming P-disks. Their co-existence requires their cooperation; and this cooperation takes two forms. First, when an application decides to discard incoming data without reading it, the system copier should (obviously) avoid copying it. An application that proactively "helps" the system in this way ends up improving the P-disk draining time of the entire system. Second, the system-level copier must exercise care not to compete

Applications	Time (*s*)
Pnapster	4.3
PwebCache1	6.6
PwebCache2	1.4
Pnapster & PwebCache1	100.3
Pnapster & PwebCache2	26.0

Table 6.5: Inter-application interference.

against application-initiated reads. In the example of Table 6.4, as an email application retrieves a large attachment, the nature of the DVD media is such that an overzealous competing system copier ends up reducing the aggregate bandwidth by a factor of nearly 10.

• *Cooperation among applications.* We have discussed the interaction between an application and the system copier when processing an incoming P-disk. We now examine interactions among applications. Although the applications are given complete control of their reads from P-disks, as observed in Section 6.3, it is important that they read what is minimally necessary and leave the rest to the system copier. Overzealous applications that "prefetch" a large amount of data from a P-disk on their own, for example, may end up harming all applications, including themselves. We consider a simple example in Table 6.5. "Pnapster" retrieves a movie trailer (13 MB) from a DVD P-disk; "PwebCache1" retrieves the entire business subsection of CNN (321 entries, 8.8 MB); and "PwebCache2" recursively retrieves all the attributes under news.yahoo.com (341 entries). When multiple of these applications are active simultaneously, we consider the time it takes all of them to finish. Again, the nature of the DVD media is such that significant interference among applications may result if they are too eager reading P-disks. Each of these applications would have been better off only satisfying an interactive user's immediate needs and letting the system background copier move data out of the P-disk.

While the results in this section are based on DVDs, we believe they are generally important for two reasons. First, practically, DVD media is a very attractive P-disk candidate, and some of its fundamental characteristics (such as latency) are likely to be with us for some time. Second, even if we were to consider other types of movable media, such as IBM Microdrive-type disks, due to energy, noise, and size considerations, these storage devices are likely to share similar issues as DVDs, so the lessons that we have learned about getting the most of DVD P-disks may be more generally applicable.

Chapter 7

Related Work

We now discuss work related to some of the major aspects of the Postmanet system.

Delay-Tolerant Networking and ad hoc Routing

Recent efforts on “Delay-Tolerant Networks” (DTNs) [15, 24, 29, 58] have started to examine the use of WiFi-enabled mobile elements (such as buses equipped with storage devices) to simulate “delayed” connectivity to places that have access to none today. While “postal classes of service” have been mentioned, to the best of our knowledge, the postal system has so far only been mentioned as an *analogy*—no existing DTN that we are aware of literally uses the postal system. There are several important differences between existing DTNs and the Postmanet. First, while existing DTNs are largely confined to relatively small regions or specialized environments, the postal system is a truly *global* “network” that reaches a far greater percentage of the world’s *human* population without needing investment in exotic equipment. Ad hoc routing, frequently a central focus of some DTNs, is not necessarily a top focus of the Postmanet. Instead, we are more concerned with somewhat less conventional routing metrics, such as the number of storage devices handled per site per postman visit.

Second, most existing DTNs are also frequently referred to as “challenged networks:” they may be limited by low bandwidth among mobile ad hoc elements, brief and/or intermittent contacts among these elements, small amounts of storage space on these nodes, and power consumption constraints. In contrast, the P-disks in the Postmanet are “dumb” and “dormant” during transit in the postal system. When they reach their destinations, they are “plugged in,” quite possibly with high-bandwidth wired alternatives (such as USB2 or Firewire). Once such “contacts” are established, they may remain connected for extended periods of time. Instead of carefully conserving resources such as storage space and bandwidth, we may in fact strive to “waste” some of these abundant resources in order to gain other advantages. Another unique aspect of the Postmanet is the possible availability of a complementary low-latency low-bandwidth Internet connection: the techniques involved in the *parallel* exploitation of multiple connectivity technologies are different from those involved in the *sequential* forwarding of data from one connectivity technology to another.

Numerous routing protocols have been proposed for ad hoc [55] and sensor networks [25, 37, 45]. Power efficiency is often a critical consideration in these networks. In a mobile ad hoc network, the routing challenge is to deliver data in the presence of unpredictable topology changes. Some protocols proactively search and cache routes to other nodes [43, 48] while others follow an on-demand approach [27, 47]. In sensor networks, often a primary focus of the routing protocol is on the “data gathering” operation—getting data from sensors to one or more base stations. The sensors and base station(s) could be fixed [25, 37, 45] or mobile [29]. Unlike ad hoc or sensor networks, the Postmanet routing challenge is not route discovery but route selection. More specifically, one can imagine that the peer nodes “connected” by the postal system constitute a “complete graph” in the sense that any node has the ability to send a P-disk to any other node. So the task we face is not to *discover* a route through a small number of neighbor nodes; instead, our task is to *select* a route in a complete graph where *all* routes are possible, subject to constraints such as end-to-end latencies and the numbers of P-disks handled per node.

The de Bruijn interconnection topology has been used in parallel applications [7, 14, 56, 59] and distributed hash tables (DHTs) [31]. These DHT-based systems employ implicit routing wherein routing decisions are made locally without requiring elaborate knowledge of the global topology. We note, however, that implicit routing may be of limited value in Postmanet, where the control and data traffic can be conveyed on different networks—the LLLB Internet could be used for dispersing topology information or topology repairs, while bulk data is communicated over the HLHB channels. In absence of an LLLB channel, however, implicit routing may again become important. A problem that has not been considered by both the parallel computing and the DHT communities is how to construct a de Bruijn graph in a geography-aware fashion for systems where communication between different pairs of nodes incurs different amount of latencies. We have devised geography-aware de Bruijn topologies for use in the Postmanet.

Active Networking, Asynchronous Communication Systems,

The use of application-specific handlers to install or update application-specific code or to copy data from an incoming P-disk to the village machine in an application-specific manner provides a great deal of flexibility to the applications. In principle, this approach is similar to prior asynchronous communication mechanisms such as Active Messages [64] and the programming languages built on top of them [10], in which handler codes associated with messages are asynchronously executed upon arrival of the messages to incorporate the newly arriving data into ongoing computations. Almost all existing active networking efforts to date [1, 12, 44, 69], however, have consciously avoided tackling persistent storage inside the network. The type and granularity of codes being distributed in a Postmanet are qualitatively different. The close collaboration that is required between application-specific data copiers and system-level data copiers also follows the precedents of a vast body of prior work in user-extensible operating systems [4, 9, 21, 30, 40, 71].

Rover is a toolkit for constructing applications targeting weak and intermittent wireless networks [28]. A key element of the system is an asynchronous communication mechanism that allows applications running on mobile wireless clients to continue to function as communication with a remote server occurs in the background. The need of an asynchronous communication mechanism applies to the high-latency Postmanet. The characteristics of the postal system, however, are different from those of a weak wireless network: the postal system provides a high-latency high-bandwidth datagram-like service. By simultaneously exploiting an available low-latency low-bandwidth Internet connection and the excess capacity of movable storage media, we can provide better higher-level services.

Using Postal System to Exchange Data

The idea of sending cassettes in the postal system has been around since the invention of the cassettes. A cassette in the postal system is sent from one person to one other person; and one cannot really send a cassette to the "entire world." In the Postmanet, when a DVD's digital content is deposited in the repository, which is accessible by the "entire network," a person literally has the power to send something to the entire world. The power of a "network effect" is much more pronounced than making point-to-point connections between people who already know each other; it is about building eBay-like communities of people who otherwise would not have connected with each other. So the repository plays an important role in realizing the "network effect" of the Postmanet, and this is why the repository abstraction is a crucial part of the systems support we provide.

Companies such as AOL and Netflix have used the postal system to deliver software and movies on a large scale. None of these existing attempts, however, have turned the postal system into a generic two-way communication channel that can cater to a wide array of applications. Gray and his colleagues have shipped via the postal system entire NFS servers filled with terabytes of astronomy data [23]. NFS servers are chosen as mobile storage devices to minimize the amount of manual configuration a data recipient would need to perform. This is a goal that we share. Our interest is in generalizing these tailor-made solutions for specialized applications into a generic communication mechanism that can benefit many applications. By itself, a local file system interface that grants application access to the mobile storage devices may be inadequate: for example, tasks such as recipients' sending back acknowledgements over the Internet should be automated away by a transport-level system. We also note that the applicability of the Postmanet approach is by no means limited to data-intensive scientific applications: we have discussed a variety of applications that can be useful for average users, especially those who fall on the wrong side of the digital divide.

The PersonalRAID system leverages a single mobile storage device that always accompanies its owner to transport storage system differences across multiple computers for a single user [61]. Coda [33, 42] is a client/server file system that allows disconnected or weakly connected mobile clients to operate out of their local "hoard." Bayou [49, 62] is a peer-to-peer application construction framework that allows its users to craft application-specific mergers and conflict resolvers for dealing with concurrent writes to the same objects. Fluid Replication [32] an extension based on Coda, introduces an intermediate level between mobile clients and their stationary servers, called “WayStations,” which are designed to provide a degree of data reliability while minimizing the communication across the wide-area used for maintaining replica consistency. These systems differ from the Postmanet in two important ways. First, these systems rely on keeping a complete copy of all the data at each location. Bayou and PersonalRAID attempt to maintain complete replicas at all locations. Coda and Fluid Replication allow a mobile client to contain only a subset of the data while the servers and WayStations should contain full replicas. In Postmanet, the data in the object repository at a hub is not meant to be housed in its entirety at any other hub location or at its client spoke locations. Doing so is neither desirable nor feasible due to resource constraints. As such, Postmanet sites have flexibility in deciding what data to keep. Second, the goal of these distributed mobile storage systems is to provide the illusion of a coherent *disk* or *file* system, while the goal of the Postmanet is to provide the illusion of a *network* connection—these are very different abstractions. The network abstraction is at a sufficiently low level that may allow potentially greater degree of application flexibility, while an important goal of typical distributed storage systems is to entirely abstract away device or machine identities. It is possible to build a Coda or Bayou-like system on top of the Postmanet in situations where complete replication is acceptable. The question of how to build a distributed storage system appropriate for our environment on top of the Postmanet, however, is still an interesting one.

Tutored Video Instruction

We draw our inspiration for the Digital StudyHall from the “Tutored Videotape Instruction” (TVI) program conducted at Stanford in the 1970s [17]. In the TVI paradigm, minimally-edited videos of unrehearsed lectures are viewed by small groups of students assisted by a *facilitator*. The facilitator is not an expert teacher; instead, the (simpler) job of the facilitator is to pause the video tape when questions arise during the play back of the taped lectures. Such questions can be asked by students who are currently viewing the tape, the instructor on the tape, or the students captured on the tape. During such a pause, the facilitator attempts to guide the students toward a resolution of the question using a discussion format, before he resumes playing the tape. When not many questions are asked, it is also the facilitator's job to instigate more questions and discussions.

Over a number of years, careful studies were performed to compare the performance of the students participating in the TVI program against that of three other groups: one was the group of students in the live classrooms; the second was the group of students who viewed the lectures live on close-circuit TV and were given the option of interacting with the instructor using phone hookups; and the third was the group of students who viewed the delayed taped lectures without any augmentation. We call these three groups the "live" group, the "interactive TV" (or ITV) group, and the "TV" group, respectively.

While it was not surprising that the TVI group outperformed the TV group, what was especially interesting was that the TVI group outperformed *all* three other groups, including the live group, consistently over the years! Indeed, the TVI students appeared to lag behind the live (on-campus) students in terms of their admission qualifications; yet the TVI mechanism appeared to have more than compensated for this initial disadvantage.

The conjecture is that there is more systematic interaction built into the TVI model than even that in the live classroom, and this increased amount of discussion in a less inhibiting environment also fosters the formation of a group discovery process that builds communication and team skills, all of which ultimately contributing to an enhanced learning process [35]. Variations of the TVI approach have been tried subsequently [13, 3], and similar degrees of success are observed.

What we can learn from these previous experiences is the following. Recorded classes, while not sufficient by themselves, when coupled with successful instigation of systematic interaction, can be extremely powerful in accomplishing education goals. Successful instigation of interactions can be effected with simple means. When successfully employed, the TVI approach unleashes the vast potential of peer-learning: students teaching students. Many “ingredients” in our Digital StudyHall are different. We are teaching very young children in rural India, not college or graduate students in the U.S. The subject matters are different. The type of interactions that we need to engage in are different. The technologies involved are different. To the best of our knowledge, we are not aware of comparable projects that address these issues. On the other hand, we believe that the lessons that we can learn from the TVI experiences dating back to the 1970s are still valid and, indeed, inspirational!

Chapter 8

Conclusion

In this thesis, we propose turning storage media transported by the postal system into a digital transport mechanism. The resulting generic asynchronous digital network, which we call the Postmanet, allows us to extend pervasive, high-bandwidth, and low-cost connectivity to places such as third world countries, where conventional networking technology is beyond people's reach today. To fully realize the potential of this approach, however, an application writer needs better support than being told to just burn discs and toss them into the postal system. In this thesis, we study the systems support that we must provide in order to achieve the generality, transparency, efficiency, and scalability goals of the Postmanet. We examine three support systems and an extensive application.

The first system, PHTTP, provides end-point support: it allows "disconnected" browser-based client applications to interact with their remote server counterparts via the Postmanet. In addition to low-level functionalities such as marshaling/unmarshaling "messages" onto/from DVDs and ensuring their authenticity, integrity, and best-effort reliability, PHTTP also provides the abstraction of a distributed object repository, a mechanism for distributing and updating application code on "disconnected" sites, and controllers for managing “DVD robots” that automate mass-processing of incoming and outgoing DVDs at hub sites. The level of sophistication involved in these system support functionalities is likely beyond what most application programmers should be required to handle on their own. The inclusion of a distributed object repository inside PHTTP that is programmatically accessible by the postal system is particularly important. It presents a service that is akin to that of a distributed file system, by making a single name space available “everywhere,” including places that lack conventional networking access, and allowing all these distributed sites to perform read, write, navigation, search and other operations on this name space. This high-level abstraction makes it easier to construct distributed Postmanet applications. The PHTTP object repository also helps realize a “network effect,” as digital messages arriving via the Postmanet are deposited in a database accessible by the “entire world.” The benefits that one can derive from this network effect are more profound than that can be derived from mere point-to-point connections.

The second system focuses on “routing,” the issue of how to get data from the sender to the receiver via the postal system. The naive approach of leaving end users to burn and swap discs on their own does not scale well, because as many as N^2 discs may need to be exchanged in an N-node network on any given day. Such an approach is slow, expensive, and requires much tedious manual labor at all sites. In conventional networking, packets are routed through a sequence of shared routers and links inside the network on their way from a sender to a receiver, so that one does not have to lay down N^2 physical links in an N-node network. Similarly, in the Postmanet, our solution to the scalable routing problem is to route Postmanet discs through a number of intermediate hops. At each of these intermediate hops, data from incoming discs is de-multiplexed, and then re-multiplexed onto outgoing discs destined to next hops. The aim is to devise a dynamic routing topology that simultaneously minimizes the number of discs that each node needs to handle and the end-to-end postal latency. In this part of the study, we present a range of routing heuristics, including those that rely on dedicated data distribution centers inside the infrastructure, those that rely on Postmanet end nodes themselves to forward data for each other, and hybrid combinations. We do not claim to have found the optimal strategies, as the theoretical formulation of this problem is likely NP-hard; but our current conjecture is that a hybrid strategy, when combined with a routing graph topology based on the de Bruijn graphs, might offer the best tradeoff.

The third system is an experimental message passing system that offers more flexibility than the browser-based PHTTP substrate described above, and it allows us to explore a number of solutions to current deficiencies in the PHTTP system. The issues addressed in this message passing system include: intelligent handling of bursty arrival of bulk data so that interactive applications are not unnecessarily blocked or delayed, exploitation of an LLLB link (such as a ham radio link) in the Postmanet for transmitting control messages (such as acknowledgements) and data (such as low-resolution versions of data otherwise transmitted on the HLHB link), support for undo operations that can "shoot down" earlier messages while they are in various intermediate in-transit stages, and intelligent exploitation of knowledge of storage media layout for performance optimization. An important insight of this exercise can be best illustrated by the relationships among the various data copiers. Application-specific data copiers can be more efficient because they alone may have the knowledge of, for example, whether certain data can be discarded without being read, or where certain data should be placed. On the other hand, system-level data copiers can be more efficient because they alone may possess low-level storage layout information and/or the opportunity to coordinate multiple user applications. These effects, though visible in conventional contexts as well, are particularly profound in the

Postmanet environment due to the large data sizes involved. For maximum efficiency, we need both system copiers and application-specific copiers to work together.

We have built and deployed a substantial real-world application, a rural distance learning system called the Digital StudyHall, on top of the Postmanet. The Digital StudyHall consists of a network of hubs and spokes, where the hubs are typically distributed in urban centers of excellence, which "radiate" contextually meaningful and coherent content and methodology into village and slum schools in their vicinity. A prototype system has been operating in rural India since July of 2005. Our experiences with the system not only have provided us insights on the systems support that we need in order to fully realize the potential of the Postmanet, but also have allowed us to conduct interesting mediation-based pedagogy studies that have proved promising in extending high-quality education to a needy population.

Future Work

To a large extent, the messaging layer that we have discussed in Chapter 6 is an experimental "playing ground" that has allowed us to quickly gain understanding of a number of issues before we fully incorporate the solutions into the production-quality PHTTP system (Chapter 4). Among the first tasks of our future work is to "graft" the validated ideas learned at the message layer onto the PHTTP system. These include a system-level data copier that can exploit its knowledge of the DVD data layout and collaborates with application-specific data copiers, exploitation of a "second network" (based on ham radios or cellular phones deployed to village schools), and undo operations.

As we have discussed in Chapter 3, our current Digital StudyHall production testbed consists of only a single hub and its spokes. While operations within a single-hub network are transparent and fully automated, and several other hubs are in various different stages of development, we do not yet have the full system support that we need to present a coherent view of the entire multi-hub network. Communication spanning multiple hubs, at the time of this writing, still requires manual intervention by each hub operator on the route. The routing strategies and algorithms discussed in Chapter 5 are only a first step toward a single system view of the entire network, and much more work needs to be done. We are continuing the investigation of routing algorithms. And to fully implement any of these routing algorithms, we need to resolve other design issues such as end-to-end naming and addressing. Our eventual aim is to build a higher-level system that, for the lack of a better term, can be described as an "education Gnutella," whose individual nodes are connected together via a variety of connectivity technologies, including the Postmanet. A human user of this education Gnutella system should not have to be aware of the disparity of such a complex and diverse network, where data is located, from which site to receive the data, and how the data is to be transmitted. Instead, the users should be able to access the system as a single logically coherent database.

Only when we achieve this goal, can we hope to succeed in realizing the "Learning eBay" vision, a vision of a coherent content repository, simultaneously acting as a meeting place, where learners and teaching staff can collaborate across time and space. We hope such a learning system will scale up to encompass a vast number of disadvantaged children, contributing toward the Millennium Development Goal of universal primary education.

Bibliography

[1] D. S. Alexander, M. Shaw, S. Nettles, and J. M. Smith. Active bridging. In *Proc. of ACM SIGCOMM '97*, pages 101–111, 1997.

[2] R. Anderson. 'Trusted Computing' Frequently Asked Questions. http://-www.cl.cam.ac.uk/˜rja14/tcpa-faq.html, August 2003.

[3] R. Anderson, M. Dickey, and H. Perkins. Experiences with tutored video instruction for introductory programming courses. In *Proc. of the Thirty-Second SIGCSE Technical Symposium on Computer Science Education*, 2001.

[4] T. Anderson, B. Bershad, E. Lazowska, and H. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. In *Proc. of the 13th ACM Symposium on Operating Systems Principles*, pages 53–79, February 1992.

[5] I. Avramopoulos, H. Kobayashi, R. Wang, and A. Krishnamurthy. Highly secure and efficient routing. *Proc. IEEE INFOCOM*, March 2004.

[6] G. Banga, P. Druschel, and J. C. Mogul. Resource Containers: A New FAcility for Resource Management In Server Systems. In *Proc. of the Third Symposium on Operating Systems Design and Implementation*, February 1999.

[7] J.-C. Bermond and P. Fraigniaud. Broadcasting and Gossiping in de Bruijn Networks. *SIAM Journal on Computing*, 23(1):212–225, 1994.

[8] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. Eggers. Extensibility, Safety and Performance in the SPIN Operating System. In *Proceedings of the ACM Fifteenth Symposium on Operating Systems Principles,* December 1995.

[9] E. Bugnion, S. Devine, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. In *Proc. of the 16th ACM Symposium on Operating Systems Principles*, pages 143–156, October 1997.

[10] D. E. Culler, A. Dusseau, S. C. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Proc. of Supercomputing '93*, November 1993.

[11] N. De Bruijn. A Combinatorial Problem. In *Proc. Koninklijke Nederlandse Akademie van Wetenschappen*, volume 49, pages 758–764, 1946.

[12] D. Decasper, Z. Dittia, G. M. Parulkar, and B. Plattner. Router plugins: A software architecture for next generation routers. In *Proc. of ACM SIGCOMM '98*, pages 229–240, 1998.

[13] J. Dutra, J. F. Gibbons, R. L. Pannoni, M. J. Sipusic, R. B. Smith, and W. R. Sutherland. Virtual Collaborative Learning: A Comparison between Face-to-Face Tutored Video Instruction (TVI) and Distributed Tutored Video Instruction (DTVI). Technical Report TR-99-72, Sun Microsystems Labooratories, January 1999.

[14] A. Esfahanian and S. Hakimi. Fault–Tolerant Routing in de Bruijn Communication Networks. *IEEE Trans on. Computers*, 34(9):777–788, 1985.

[15] K. Fall. A delay tolerant networking architecture for challenged internets. In *Proc. of ACM SIGCOMM 2003*, August 2003.

[16] N. Garg, S. Sobti, J. Lai, F. Zheng, K. Li, A. Krishnamurthy, and R. Wang. Bridging the digital divide: storage media + postal network = generic high-bandwidth communication. *ACM Transactions on Storage*, Volume 1(2), May 2005.

[17] J. F. Gibbons, W. R. Kincheloe, and K. S. Down. Tutored Videotape Instruction: A New Use of Electronics Media in Education. *Science*, 195(3), 1977.

[18] G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File Server Scaling with Network-Attached Secure Disks. In *Proc. of the 1997 SIGMETRICS*, June 1997.

[19] Gnutella. http://gnutella.wego.com/.

[20] A. Goldberg. Network Optimization Library. http://www.avglab.com/andrew/soft.html.

[21] D. Golub, R. Dean, A. Forin, and R. Rashid. Unix as an Application Program. In *Proc. of the 1990 Summer USENIX*, June 1990.

[22] Graph Theory and Combinatorics Research Group, Applied Mathematics IV Department, Universitat Politcnica de Catalunya (UPC). The (Degree,Diameter) Problem for Vertex-symmetric Digraphs. http://www-mat.upc.es/grup_de_grafs/grafs/taula_vsd.html.

[23] J. Gray and D. Patterson. A Conversation with Jim Gray. *ACM Queue*, 1(4), June 2003.

[24] A. A. Hasson, R. Fletcher, and A. Pentland. DakNet: A Road To Universal Broadband Connectivity. http://courses.media.mit.edu/2003fall/de/DakNet-Case.pdf, 2003.

[25] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol forwireless microsensor networks. In *Hawaii International Conference on System Sciences*, January 2000.

[26] J. B. Horrigan. Adoption of Broadband to the Home. The Pew Internet and American Life Project, http://www.pewinternet.org/reports/toc.asp?Report=90, May 2003.

[27] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[28] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and M. F. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *Proc. the 15th ACM Symposium on Operating Systems Principles*, pages 156–171, December 1995.

[29] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *The Tenth International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[30] M. F. Kaashoek, D. R. Engler, G. R. Ganger, H. M. Briceno, R. Hunt, D. Mazieres, T. Pinckney, R. Grimm, J. Jannotti, and K. Mackenzie. Application Performance and Flexibility on Exokernel Systems. In *Proc. the 16th ACM Symposium on Operating Systems Principles*, pages 52–65, October 1997.

[31] M. F. Kaashoek and D. R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proc. of Intl. Workshop on Peer-to-Peer Systems*, 2003.

[32] M. Kim, L. P. Cox, and B. D. Noble. Safety, Visibility, and Performance in a Wide-Area File System. In *Proc. First Conference on File and Storage Technologies*, January 2002.

[33] J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems*, 10(1):3–25, Feb. 1992.

[34] A. Krishnakumar. The decline of public education. *Frontline*, 21(16), July 2004.

[35] E. Lazowska. UW CSE Tutored Video Pilot Project. http://www.cs.washington.edu/education/TVI/gibbons.html.

[36] A. Lenhart, J. Horrigan, L. Rainie, K. Allen, A. Boyce, M. Madden, and E. O'Grady. The ever-shifting internet population: A new look at internet access and the digital divide. http://www.pewinternet.org/report_display.asp?r=88, April 2003.

[37] S. Lindsey, C. Raghavendra, and K. Sivalingam. Data gathering in sensor networks using the energy*delay metric. In *Proceedings of the Parallel and Distributed Processing Symposium*, April 2001.

[38] MacWorld. Sony is developing 200GB Blu-ray storage. http://www.macworld.com/news/2004/09/21/sony/index.php.

[39] M. Madden. America's Online Pursuits: The changing picture of who's online and what they do. The Pew Internet and American Life Project, http://www.pewinternet.org/reports/toc.asp?Report=106, December 2003.

[40] H. Massalin and C. Pu. Threads and input/output in the synthesis kernel. *Operating Systems Review*, 23(5):191–201, 1989.

[41] K. S. McNamara. Information and Communication Technologies, Poverty and Development: Learning from Experience. In *A Background Paper for the infoDev Annual Symposium, The World Bank*, December 2003.

[42] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the ACM Fifteenth Symposium on Operating Systems Principles*, December 1995.

[43] S. Murthy and J. J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications*, 1(2):183–197, 1996.

[44] E. L. Nygren, S. J. Garland, and M. F. Kaashoek. PAN: A High-Performance Active Network Node Supporting Multiple Mobile Code Systems. In *Proc. of OpenArch'99*, March 1999.

[45] C. M. Okino and M. G. Corr. Statistically accurate sensor networking. In *Proceedings of Wireless Communication and Networking Conference*, March 2002.

[46] OpenLDAP Foundation. Openldap 2.1. http://www.openldap.org.

[47] C. Perkins. Ad-hoc on-demand distance vector routing, 1997.

[48] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[49] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proc. the 16th ACM Symposium on Operating Systems Principles*, pages 288–301, October 1997.

[50] The Postman Always Rings Twice. http://www.cs.princeton.edu/˜rywang/distance, 2004.

[51] C. K. Prahalad and S. L. Hart. The Fortune at the Bottom of the Pyramid. *strategy+business*, issue 26, 2002.

[52] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM 2001*, pages 161–172, August 2001.

[53] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. In *Proc. of the 13th Symposium on Operating Systems Principles*, pages 1–15, Oct. 1991.

[54] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, November 2001.

[55] E. Royer and C. Toh. A review of current routing protocols for ad-hoc mobile wireless networks, 1999.

[56] M. Samatham and D. Pradhan. The de Bruijn Multiprocessor Network: A Versatile Parallel Processing and Sorting Network for VLSI. *IEEE Trans on. Computers*, 38(4):567–581, 1989.

[57] Scientific American. More Bits in Pits. In *Scientific American*, February 2005.

[58] R. Shah, S. Roy, S. Jain, and W. Brunette. Data mules: Modeling a three-tier architecture for sparse sensor networks. In *IEEE SNPA Workshop*, May 2003.

[59] K. Sivarajan and R. Ramaswami. Multihop Lightwave Networks based on de Bruijn Graphs. In *Proc. INFOCOMM*, pages 1001–1011, 1992.

[60] Sleepycat Software. Berkeley db 4.2. http://www.sleepycat.com.

[61] S. Sobti, N. Garg, C. Zhang, X. Yu, A. Krishnamurthy, and R. Y. Wang. PersonalRAID: Mobile Storage for Distributed and Disconnected Computers. In *Proc. First Conference on File and Storage Technologies*, January 2002.

[62] D. B. Terry, M. M. Theimer, K. Peterson, A. J. Demers, M. J. Spreitzer, and C. H. Hauaser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proc. the 15th ACM Symposium on Operating Systems Principles*, pages 172–183, December 1995.

[63] The United Nations. UN Millennium Development Goals. http://www.un.org/millenniumgoals/.

[64] T. von Eicken, D. Culler, S. Goldstein, and K. E. Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 256–266, May 1992.

[65] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient Software-Based Fault Isolation. In *Proceedings of the ACM Fourteenth Symposium on Operating Systems Principles*, December 1993.

[66] R. Wang, U. Sahni, S. Sobti, N. Garg, J. P. Singh, M. Kam, A. Krishnamurthy, and T. Anderson. The Digital StudyHall. Technical Report TR-723-05, Computer Science Department, Princeton University, http://dsh.cs.washington.edu:8000/distance/studyhall.pdf, March 2005.

[67] R. Y. Wang, N. Garg, S. Sobti, J. Lai, E. Ziskind, F. Zheng, A. Nakao, and A. Krishnamurthy. Postmanet: Turning the Postal System into a Generic Digital Communication Mechanism. In *Proc. ACM SIGCOMM 2004*, August 2004.

[68] R. Y. Wang, N. Garg, S. Sobti, J. Lai, E. Ziskind, F. Zheng, A. Nakao, and A. Krishnamurthy. Postmanet: Turning the Postal System into a Generic Digital Communication Mechanism. Technical Report TR-691-04, Computer Science Department, Princeton University, February 2004.

[69] D. Wetherall. Active network vision and reality: lessons from a capsule-based system. In *Proceedings of the ACM Seventeenth Symposium on Operating Systems Principles*, December 1999.

[70] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, February 1994.

[71] W. A. Wulf, E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson, and F. Pollack. HYDRA: The Kernel of a Multiprocessor Operating System. *Communications of the ACM*, 17(6):337–345, June 1974.

[72] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2004.