A Network Architecture for Heterogeneous Mobile Computing

DRAFT

Eric A. Brewer, Randy H. Katz, Elan Amir, Hari Balakrishnan, Yatin Chawathe, Armando Fox, Steven D. Gribble, Todd Hodes, Daniel Jiang, Giao Nguyen, Venkata N. Padmanabhan, Mark Stemm, Tao Ye

October 14, 1998, Version 2 (incomplete)

This document covers the design and architecture of the Dædalus/Glomop network infrastructure, which provides optimized network access to heterogeneous clients in heterogeneous and potentially overlapping networks. The architecture comprises four components that provide orthogonal, but complementary, functionality at all major points in the traditional layered network model: wireless overlay networking, transport layer performance optimizations, a scalable proxy (middleware) subsystem, and application-level network services. We show how the interaction of these components enables "anytime, anywhere" wireless access that is easy to use, delivering the best performance available at any given time while adapting intelligently to a wide range of network conditions and client device characteristics. We describe our progress to date in implementing the architecture, as well as what we see as remaining milestones and future work.

Online project information and publications: ***http://daedalus.cs.berkeley.edu***

1 Introduction 7
1.1 Routing to Mobile Hosts: Overlay Networking 8
1.1.1 Overlay Networking Details 10
1.2 Transport Performance: Optimizing the Network 12
1.3 Scalable Proxy 12
1.4 Network Services and Authentication 13
1.5 Key Assumptions (What Aren’t We Doing) 13
1.6 Summary 14

2 Overlay Networking 15
2.1 Overview 15
2.2 Triggering Handoffs 16
2.3 Techniques for Low-Latency Vertical Handoffs 18
2.3.1 Hints for Enabling Enhancements 18
2.3.2 Enhancements 19
2.4 Differences between Vertical Handoff system and IETF Mobile IP 20
2.5 Overlay Networking Summary 21

3 Reliable Data Transport Performance 23
3.1 The Snoop Protocol 24
3.2 Asymmetric Networks 26
3.3 Link Sharing for Wireless LANs 29
3.4 Bandwidth Sharing 29
3.4.1 Surplus Bandwidth Distribution 31
3.5 Summary 32

4 The Scalable Proxy and Coordination Bus 33
4.1 The Coordination Bus 33
4.2 The Proxy Architecture 34
4.2.1 The Proxy Front End 34
4.2.2 Workers 35
4.2.3 Proxy Transcoder Manager (PTM) 36
4.2.4 Caching 37
4.2.5 Video Gateway 37
4.2.6 Control Panel 37
4.2.7 Dynamic Adaptation 38
4.3 Application, Transport, and User Interface Issues 38
4.4 Summary 40

5 Network Services 41
5.1 Controllable Objects 42
5.1.1 Aggregation 42
5.1.2 Naming 43
5.1.3 Shared Control Conflicts 43
5.1.4 Cameras as Object Interfaces 44
5.2 Service Advertisement and Discovery 44

5.3 Mapping Client Controls to Exported Objects 45
5.3.1 Transduction Protocols 45
5.3.2 Complex Behaviors 45
5.4 Implementing Service Interaction 46
5.4.1 Basic Operation 46
5.4.2 System Setup 46
5.4.3 Message-level Detail 48
5.4.4 Client Bootstrap 48
5.4.5 Beaconing 49
5.4.6 Charon: Proxied Secure Access 50
5.4.7 Scoped Access Control 51
5.4.8 Client Interfaces 52
5.4.9 Naming Scheme 53
5.5 Prototype Mobile Services 54
5.5.1 Maps 54
5.5.2 Proxy and Gateway Autoconfiguration 55
5.5.3 Location Tracking 56
5.5.4 Printer Access 56
5.5.5 Motorized Cameras 56
5.5.6 405 Soda Room Interaction 57
5.5.7 326 Soda Room Interaction 59
5.6 Discussion 59
5.7 Related Work 60
5.8 Continuing Work and Future Directions 61
5.8.1 Wide-area issues 61
5.8.2 Building control and support 61
5.8.3 Delegating operations 62
5.8.4 Queued RPC 62
5.8.5 Maps 62
5.8.6 Interface specification grammar and compiler 62
5.8.7 Fault tolerance 62
5.8.8 Geographic locality 62
5.8.9 Multimedia collaboration control architecture 63
5.8.10 Conference control primitives for lightweight sessions 63
5.9 Conclusions 64

6 The Network Stacks 65
6.1 The Basestation Network Stack 65
6.1.1 Black-Box Basestations 65
6.1.2 Dædalus-Aware Basestations 65
6.2 The Client Network Stack 67
6.3 The Proxy Network Stack (for Clients) 68
6.4 Network Stack Summary 68

7 Summary 71
7.1 The Principles Revisited 71
7.2 Summary 73

8 Glossary 75

9 References 79

1 Introduction

> *"People and their machines should be able to* ***access*** *information and communicate with each other* ***easily*** *and* ***securely****, in* ***any medium*** or combination of media— *voice, data, image, video, or multimedia —* ***any time****,* ***anywhere****, in a* ***timely****,* ***cost-effective*** *way."*
>
> *Dr. George H. Heilmeier*
> *IEEE Communication*
> *October 1992*

More than any other, this statement captures the broad goals of our project and the resulting architecture that is the subject of this document. The highlighted words carry tremendous consequences that form the driving principles of this work — in fact, we will derive the principles from these words, and the architecture from these principles.

Access Our mantra is "Access is the Killer App", by which we mean that the key to mobile computing and portable devices — PDA's, smart phones, and laptops — is not an application per se, but rather access to both my desktop environment (at home or office) and to the internet at large.

Anytime, Anywhere The infrastructure required for this access must be permanently deployed and readily available from everywhere. Everywhere includes indoors, outdoors, in cities and in remote regions.

Principle 1: **Heterogeneous Networks***: The infrastructure must include wireless networks that have a mixture of global and indoor coverage, thus requiring a heterogeneous collection of networks.*

To optimize network performance, we must use support a variety of networks and pick the best among them for the current location. We call this *Overlay Networking*, since a set of networks are overlaid onto the same physical area.

Principle 2: **Scalable***: The infrastructure must scale to support millions of users.*

Principle 3: **Highly Available***: The infrastructure must be available all of the time.*

Easily It is not enough to have access; that access must be simple to use.

Principle 4: **Transparent Access***: The detection and setup of a network connection should be automatic. Users shouldn't have to know what networks are in range.*

Principle 5: **Localized Service***: The detection and setup of local network services should be automatic. Users shouldn't have to know what services are available at their current location.*

Securely Access to your own stuff requires authentication. Access to local resources may also require authentication, since not all visitors are treated the same.

Principle 6: ***Global Authentication****: We must authenticate users using a globally available security infrastructure, such as public-key cryptography or Kerberos.*

Any Medium We must deal with a wide variety of data.

Principle 7: ***Multimedia****: The infrastructure must support graphics, audio and video in addition to text.*

Timely The performance should be the best possible (for the current location).

Principle 8: ***Performance****: The user's data should arrive as fast as possible. This includes selecting the best network, optimizing the network performance, and optimizing the content at the application level.*

Cost Effective The infrastructure should be designed to minimize the costs and share and amortize resources as much as possible.

Principle 9: ***Heterogeneous Client****s: Complexity should be pushed into the infrastructure, where it can be amortized over all of the active users. The infrastructure should support both inexpensive client devices, such as smart phones, and more sophisticated computers, such as high-end laptops.*

Even with some network-level optimizations, there will be times the network performance is poor and we need application-level support. In particular, we would like to optimize the data sent to reduce the demands on the client and network without reducing the information content.

Principle 10: ***Dynamic Adaptation****: The data sent to the user should be optimized for timeliness, carrying the most information in the least amount of time. The nature of this adaptation depends on the current network, the preferences of the user, and the nature of the data (text is much different than graphics).*

These ten principles are driving forces behind the architecture. In the next four sections, we look at the key elements of the architecture and see how they meet these principles. These elements are largely orthogonal; they can be used independently of one another, but they do work well together.

1.1 Routing to Mobile Hosts: Overlay Networking

Because we must deal with several overlapping networks to achieve the best performance, we need a way to transparently switch networks. Furthermore, the best available network changes due to mobility and congestion. Thus the first task of the architecture is simply to route packets to a *mobile host* using the best available network. The overlay networking subsystem deals with the detection of available networks, the selection of the best network, and the transparent switch to either a another cell in the same network, or to a completely new network.

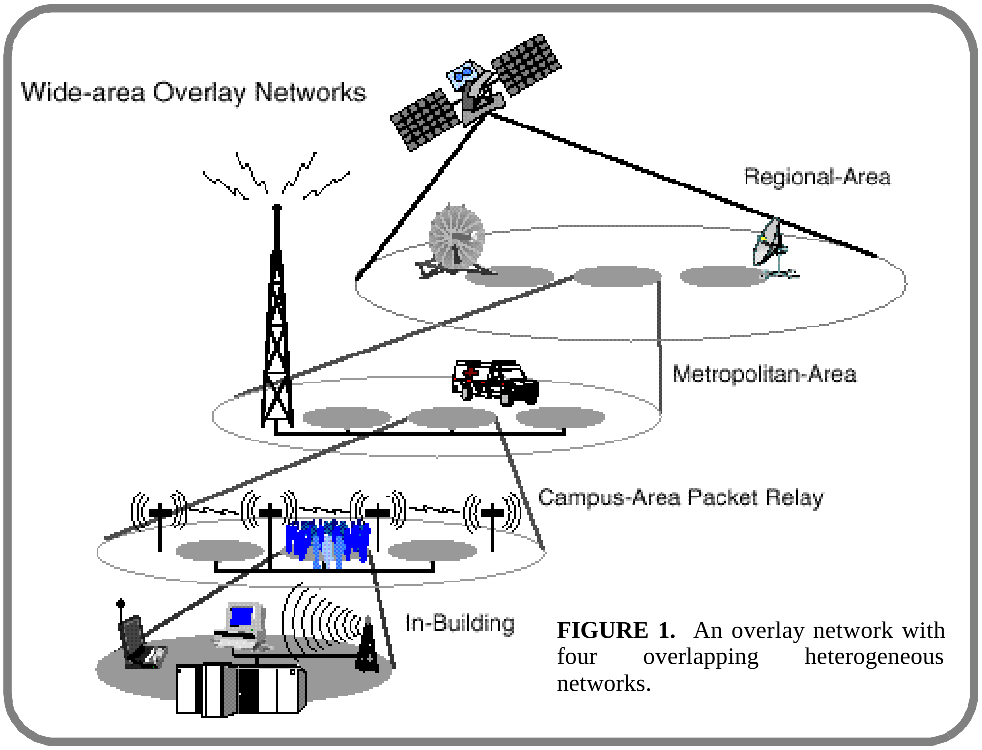


FIGURE 1. An overlay network with four overlapping heterogeneous networks.

Figure 1 shows a series of overlapping networks. The mobile host can be in any of the ovals, and when in an oval it is typically also in an oval for each of the networks above it. For example, a mobile host in a cell of an in-building network is typically also in a cell for each of the campus-area, metropolitan-area and regional networks. The mobile host can move among these cells either horizontally (within a network) or vertically (between networks); a cell transition is called a *handoff*, and overlay networking thus supports both vertical and horizontal handoffs.

Thus, the two key goals of overlay networking are to detect and monitor the currently available networks and to manage handoffs. Thus overlay networking supports the principles of **Heterogeneous Networks** and **Transparent Access**. The former comes by definition, since it assume multiple overlapping heterogeneous networks, and the latter comes from the detection of available networks.

Overlay networking also helps with several other principles:

- **Highly Available**: by exploiting multiple networks, we achieve redundancy and thus improve availability of connectivity.
- **Localized Services**: the use of multiple networks, especially those with small cells, such as IR, helps the infrastructure with the location of the client. This in turn enables accurate localized services. Note that just knowing which networks are available provides location information.
- **Performance**: overlay network improves performance by selecting the best available network.
- Dynamic Adaptation: by knowing the current network used by a client, overlay networking helps with the parameters for dynamic adaptation, i.e., we can more

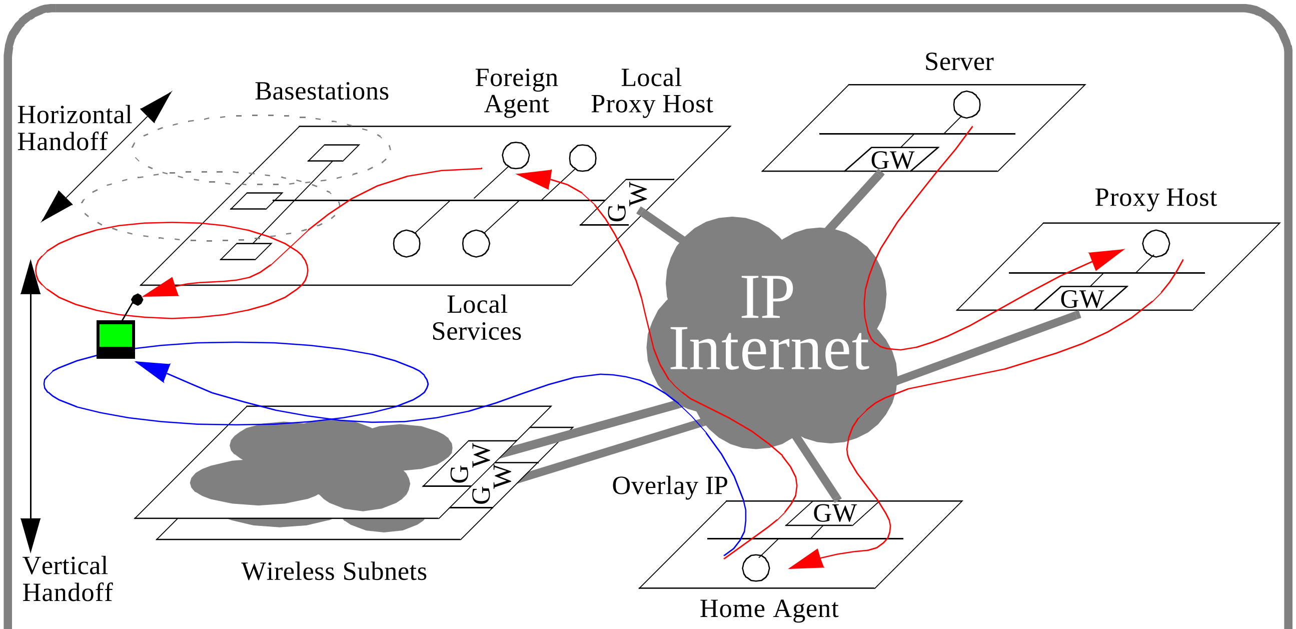


FIGURE 2. The basic system architecture has four components above the network level. In addition to clients and servers, *basestations* provide wireless connectivity, the *home agent* manages mobility and tracks the client, and *proxies* provides content optimization and security.

accurately tune the content for a user because we know the properties of their network connection.

Given this high-level description of overlay networking, we now look a level deeper and explore the mechanisms of used.

1.1.1 Overlay Networking Details

Figure 2 shows the basic architecture. The mobile host is on the left and the server is at the top right. There are three overlaid networks shown, each as a horizontal plane; horizontal handoff occurs between basestations on the same plane (network), while vertical handoff occurs between the planes.

There are two kinds of basestations in practice: *black-box basestations*, those over which we have no control, and *Dædalus-aware basestations*, we can influence enough to run our protocols. This distinction is important since we want to exploit existing deployed networks over which we have essentially no control. Typically, black-box basestations have proprietary protocols for network detection and horizontal handoff. We can normally exploit the network detection stuff for our purposes, but control of handoffs is usually out of our control. Thus, black-box basestations typically have the following responsibilities:

- Provide routing to client
- Manage local wireless network

With Dædalus-aware basestations, we have sufficient control such that we can control handoffs fairly precisely; they have the following responsibilities:

- Provide beacon to client
- Optimize network against wireless losses

- Provide location information
- Bootstrap name server for new clients in their cell

The home agent is the official (non-mobile) location of the client. It keeps track of the actual location and forwards packets to the current location of the client. Route optimization can be used to avoid “triangle routing” through the home agent. Before such an optimization, all traffic must go through the home agent. The key responsibilities of the home agent are to:

- Track client’s position and current IP address
- Forward packets to current location
- Set up optimized routes

These responsibilities mean that the server and proxy*do not* have deal with mobility.

Servers are unmodified (legacy) servers on the internet. It is also possible to integrate some of the proxy responsibilities into the server.

The proxy, which is covered in more detail below, performs dynamic adaptation for the client of the servers content. The proxy provides an indirect interface between clients and servers, thus hiding network problems and changes from servers and allowing network and content optimization for the client. Many clients may connect to a single proxy. Its responsibilities include:

- Provide optimized access to servers
- Manage connection to client
- Provide services and name server
- Optimize data formats
- Monitor network connectivity
- Enable end-to-end security for simple clients
- Track clients’ location

Finally, there are some local services associated with the (relatively small) region covered by each basestation. For example, there might a local information web server or a local printer that is available to users of the corresponding basestation.

The basic data flow is shown by the arrows. A client request, say an HTTP request, moves from the client to the basestation to the proxy. The proxy will decide what do with it, but we’ll assume that if forwards the request to the server. The sends the response to the proxy, which then sends it to the Home Agent, since it doesn’t know where the mobile host is located.[1] The Home Agent always knows where the client is, so it forwards the data to the client through the appropriate network.

Handoffs thus require updating only the Home Agent (and the involved basestations). Handoffs may occur either because the primary network becomes unavailable, or because a better network has recently become available. Users may also force handoffs explicitly.

Network detection in Dædalus-aware basestations is performed by broadcasting periodic beacons, which clients use to detect the basestation. Thus the availability of a basestation is indicated by the reception of the beacon, and handoff depends on the measured quality of the multiple network connections.

[1] As an optimization, the proxy can keep track of the last known location as a hint. This is similar to “route optimization” in Mobile IP.

At a high level, this a fairly complete picture of the architecture. The three major missing pieces are the optimizations of the transport layer, the proxy, and available services. The next three subsections cover these pieces.

1.2 Transport Performance: Optimizing the Network

There are many challenges presented by a heterogeneous collection of networks that together provide wide-area wireless access anytime, anywhere. The primary problems are low bandwidth, high error rates, asymmetric performance, and high latency. We have extended TCP to mitigate all of these issues. We also multiplex many short connections to the same place over one long-lived connection, called a “session”, which improves the performance of the connections.

Thus, the Transport Performance subsystem primarily supports the principle of **Performance**. It also supports several other principles:

- **Highly Available**: The improved error handling increases the availability of the network.
- **Multimedia**: By allowing the application to indicated its requirements through delivery classes, the architecture allows better performance for multimedia data. For example, we can avoid retransmitting lost audio or video packets, since they would likely arrive too late to be useful and the rendering of the data will probably be OK without these packets.
- **Dynamic Adaptation**: Because we can monitor the connection between the client and the proxy, we can tune the content based on the actual bandwidth, thus maximizing the effectiveness of the transfer.

Although transport performance is largely an optimization in this architecture, the wide variety of networks and network problems that we encounter indicates that these issues are not merely an optimization, but required parts of practically useful system.

1.3 Scalable Proxy

A major component of the architecture is the scalable proxy, which provides dynamic adaptation of data sent to the user based on the user’s device, network connectivity and preferences. It thus provides timely data despite the limitations of the network, and acts as intermediary between client and server, thus hiding any disconnections or client limitations.

The proxy manages all of the data seen by the client and uses caching to improve performance. Thus, clients ask the proxy for a particular object and the proxy will get it either from the cache or from the internet. The proxy also handles authentication in order to provide access to restricted resources (such as a user’s own e-mail).

The proxy is the key method by which we meet the goals of **Heterogeneous Clients** and **Dynamic Adaptation**. In particular, since heterogeneous clients can’t all handle standard servers (e.g. HTML), the proxy transforms the data into formats that the client can handle. The proxy also keeps track of user preferences and current network conditions in order to perform dynamic adaptation in a way that increases the overall **Performance** of the system and supports **Heterogeneous Networks**.

The proxy is also a key part of the **Scalable**, **Highly Available** architecture. It is designed to be fault tolerant to scale to thousands of simultaneous users. By amortizing its resources over all of the active users (but none of the inactive users), the proxy also leads to a more **Cost Effective** architecture.

Finally, the proxy supports **Multimedia** (through translations for image, PostScript and video), **Localized Services** (with name service), **Global Authentication** (by helping to authenticate impoverished clients), and **Transparent Access** (by isolating mobility and connectivity problems from legacy servers).

1.4 Network Services and Authentication

Finally, we provide mechanisms for the management and discovery of local resources, thus making the network easier to use well. In addition to automatically selecting the best available network, we provide a systematic way to automatically discover local resources such as maps or printers that increase the value of being connected.

For restricted resources and services, we need authentication to enable restricted access. We have designed and implemented an authentication system based on Kerberos [SNS88] that allows inexpensive mobile devices to authenticate themselves end-to-end with the help an untrusted infrastructure. This resolves the conflicting goals of global authentication with low-cost mobile clients that can't run Kerberos locally and don't want to trust the local infrastructure with their credentials.

The Network Services subsystem is the key to the principles of **Localized Service** and **Global Authentication**. It allows registration of local services and then provides those services to local clients. The authentication work allows users to access any Kerberos resource whether local or remote. It supports **Heterogeneous Networks** and **Heterogeneous Clients** by allowing the variations to affect the available services. For example, the service that allows you to see who else is in your cell clearly depends on the network in use.

The subsystem is **Scalable** and will be **Highly Available** in the near future. It supports **Multimedia** by allowing a variety of local servers. Finally, it enables Transparent Access by automatically determining the available services and by help clients find local proxies and other key services such as SMTP and DNS.

1.5 Key Assumptions (What Aren't We Doing)

Although this architecture is fairly complete vertical attack on the issues of heterogeneous mobile computing, there are several areas that we do not innovate and instead limit ourselves to what is already available.

Networks

We do not develop any new networks. Although all of the available wireless networks have shortcomings, developing a new network is substantial work and is not really within our core knowledge base. Instead we take several networks off of the shelf, although we may modify the link and transport layers.

Client Hardware

We limit ourselves to existing client hardware and operating systems. We do modify the networking and application software for the clients, but even that we prefer to avoid when possible. Section 4.3 has a discussion of the benefits of modifying the client software to use our application support layer (ASL).

Servers

We assume legacy servers for HTTP, FTP, news and e-mail. Many of the benefits of the proxy could be merged into servers, but we do not do so, and even if we did we could not affect a significant fraction of the servers.

Principle	Overlay Networking	Transport Performance	Scalable Proxy	Network Services
Heterogeneous Networks	Required	Helps	Helps by adapting content	Affects Services
Scalable	▨	▨	Yes	Yes
Highly Available	Multiple networks	Better Error Handling	Yes	Not Yet
Transparent Access	Required	▨	Supports	Supports
Localized Service	Helps	▨	Supports	Required
Global Authentication	▨	▨	Supports	Required
Multimedia	▨	Delivery Classes	Yes	Local Servers
Performance	Pick Best Network	Optimize Network	Optimize Content	▨
Heterogeneous Clients	▨	Assume TCP in most clients	Required	Affects Services
Dynamic Adaptation	Helps	Provides feedback on network quality	Required	▨

TABLE 1: Principles versus the Key Architecture Elements
(Gray boxes indicate *Not Applicable*)

1.6 Summary

Table 1 shows how these five elements support the driving principles. Because the elements are orthogonal, you can determine from the table which elements are required for each goal. Note that some elements help a goal but are not required to meet it. For example, the Overlay Networking module helps with Localized Service by broadcasting the IP address of the local name server used to discover services. The services could be provided without this help if the clients used some well-known server to get the address.

We have provided an overview of principles and subsystems of the architecture. The next four sections cover each of the major subsystems in more detail; they are intended to be fairly independent, so that they may be read in any order or not at allSection 6 combines all of the modules from the network stack point of view, which helps to understand how the subsystems interact. We close with a summary, glossary and a list of references.

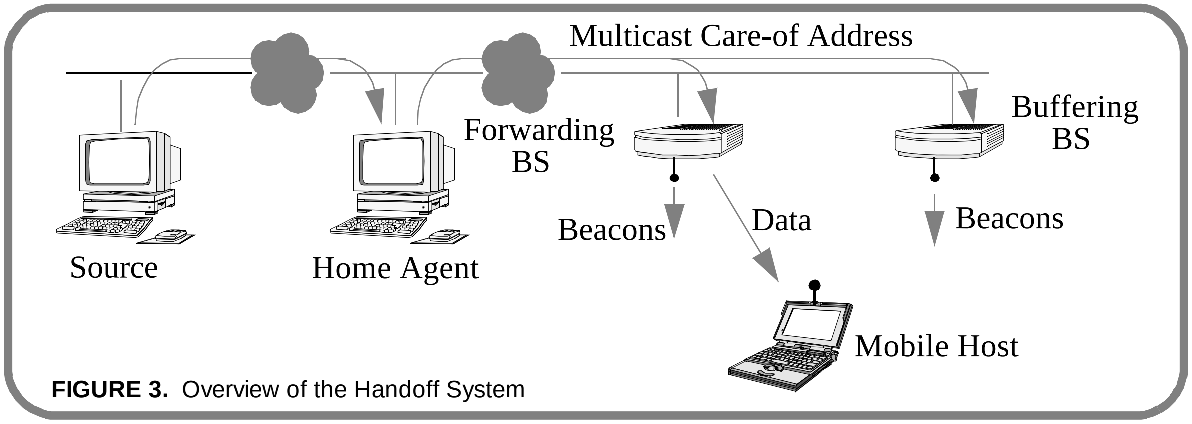


FIGURE 3. Overview of the Handoff System

2 Overlay Networking

In this section we describe the work that our group has done in implementing mobility support in Overlay Networks. This includes a *Vertical Handoff* system that enables mobility between heterogeneous wireless networks. We present an overview of the basic system, enhancements to the basic system for low-latency handoffs, and point out differences between our system and IETF Mobile IP.

2.1 Overview

The handoff system is built on top of existing mobile routing capabilities which enhance the Mobile IP specification [Per96] with a multicast-based one [Ses95] designed for low latency handoffs. Our work extends this enhanced multicast-based implementation by incorporating support for multiple wireless network interfaces [Ste97]. As shown inFigure 3, Mobile Hosts (MHs) connect to a wired infrastructure via Base Stations (BSs) which act as Foreign Agents (FAs). A Home Agent (HA) performs the same functions as in Mobile IP, encapsulating packets from the source and forwarding them to the FAs. One important difference in our implementation is that the care-of address is a multicast rather than unicast address. A small group of BSs are selected by the mobile to listen to this multicast address for packets encapsulated and sent by the HA. One of the BSs is selected by the MH to be a *forwarding* BS; it decapsulates the packets sent by the HA and forwards those packets to the MH. The other BSs are *buffering* BSs; they hold a small number of packets from the HA in a circular buffer. When the mobile initiates a handoff, it instructs the old BS to move from forwarding to buffering mode, and the new BS to move from buffering to forwarding mode. The new BS forwards the buffered packets that the mobile has not yet received. For networks in which the BS infrastructure is not under our control, the Home Agent acts as the BS to the Mobile Host; the FA functionality with respect to that wireless network is incorporated at the HA machine instead of being incorporated at the gateway between the wired and wireless network.

BSs send out periodic beacons similar to Mobile IP foreign agent advertisements. The MH listens to these packets and determines which BS should forward packets for the mobile, which BSs should buffer packets in anticipation of a handoff, and which BSs should belong to the multicast group assigned for a single mobile.

Figure 4 shows a detailed breakdown of the state and agents that implement the handoff system. The network layer of the Home Agent includes a translation table that maps from a MH’s home address to a multicast care-of address. All incoming packets are compared against the entries in the table. Matching packets are encapsulated and forwarded using the corresponding multicast care-of

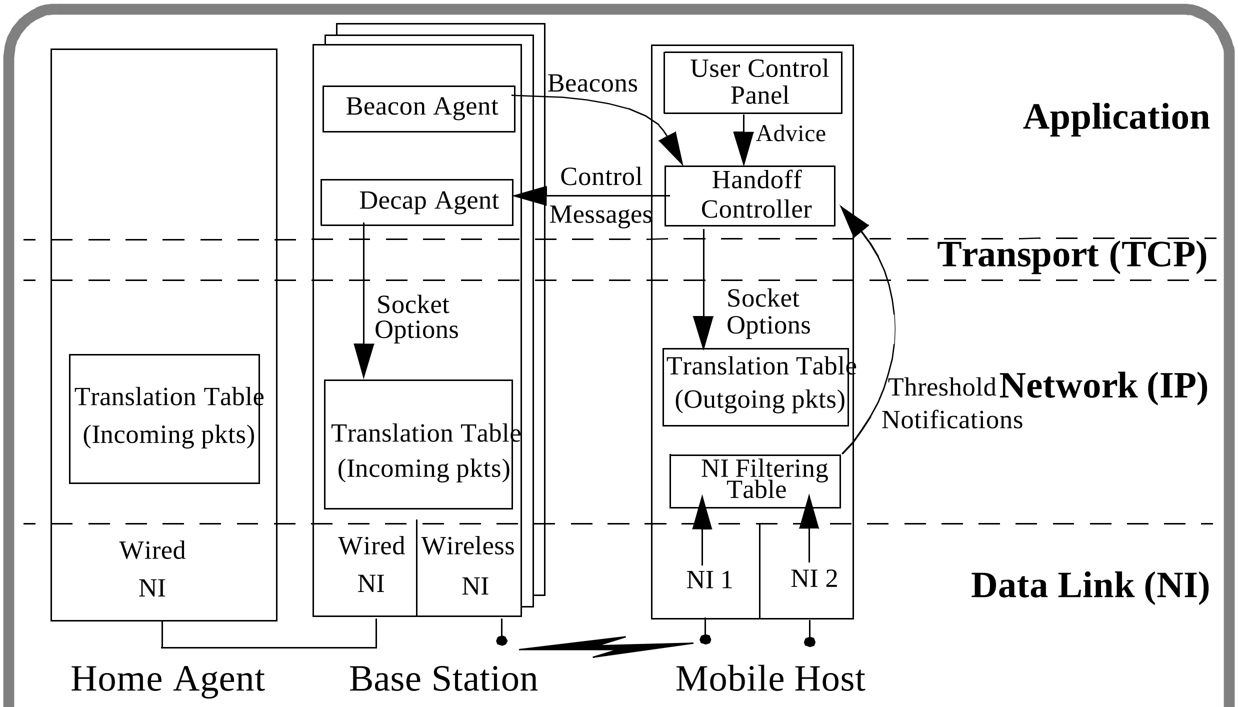


FIGURE 4. Network Stack for Home Agent, Base Station, and Mobile Host

address. At each BS there is a translation table that maps a MH's multicast care-of address to a local address. The translation table also includes the state of the BS with respect to this MH (e.g. buffering packets, forwarding packets, etc.). All incoming packets are compared against the entries in the table and the operation in the table (forward to mobile, buffer packet for mobile, etc.) is performed for matching packets. There are two user-level agents at the BS: a *beacon agent* that transmits beacon packets, and a *decapsulation agent* that receives control messages from the MH that modify the kernel-level translation table. The decapsulation agent manipulates the translation tables from user-level using socket options. At the mobile host, there is a single translation table that inserts the MH's home address in all outgoing packets. There is also a network interface (NI)-specific table that keeps track of the number of packets that have arrived for the MH over each network interface and filters out duplicate packets that are received over multiple network interfaces. A user-level process can register a callback with the networking stack to be notified when changes occur in this table. When more than a threshold number of packets arrives over a single interface, the user-level process is notified. This table and the associated threshold notification callbacks are used in the doublecasting schemes described in Section 2.3. There are two user-level agents at the mobile host: a *handoff controller* that uses beacons to determine the overlay network and BS to connect to, and a *user control panel* that allows the user to control the choice of network or BS to use via *advice*, described in [Ste97].

2.2 Triggering Handoffs

In a network of homogeneous BSs, the relative signal strength of beacons is compared and the BS with the highest is chosen as the forwarding BS. Figure 5(a) shows in detail the breakdown of a horizontal handoff. The three vertical lines represent the old BS, the MH, and the new BS, respectively, and the arrows represent messages sent from one machine to another. The BSs transmit infrequent beacon packets to the broadcast address of the local subnet. Data packets are also for-

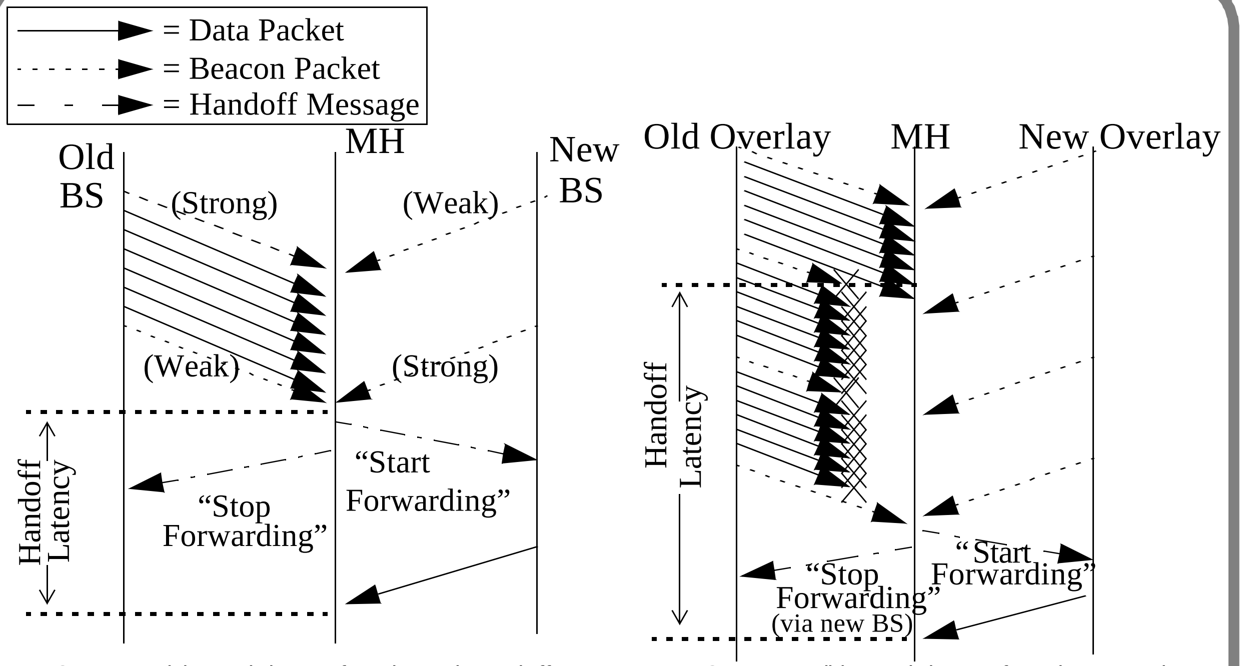


FIGURE 5. (a) Breakdown of Horizontal Handoff

FIGURE 5. (b) Breakdown of Basic Upward Vertical Handoff

warded from the old BS. At some point, the signal strength of the new BS is greater than that of the old BS, and the MH initiates a handoff to the new BS. It instructs the new BS to stop buffering packets and start forwarding packets to the MH. The MH also instructs the old BS to stop forwarding packets and start buffering packets. In the homogeneous handoff system, the handoff latency is measured from the time the mobile decides that the new BS has a larger signal strength until the first data packet arrives from it.

In our system, while a MH roams within the cells that comprise a single overlay, handoffs happen just as in the original system. The MH uses a channel-specific metric to compare different BSs and connects to the best one according to that metric. This allows the horizontal handoff system to operate seamlessly underneath the vertical handoff system. For an overlay network that handles mobility directly (for example, CDPD [CDPD] or Metricom's Ricochet [Ricochet] network), our system does nothing and lets that network make all mobility decisions.

Figure 5(b) shows the breakdown of a typical vertical handoff. An upward handoff is initiated when several beacons from the current overlay network are not received. The MH decides that the current network is unreachable and hands over to the next higher network. Even though the MH cannot directly hear the old overlay network, it must still instruct the BS of the old overlay to stop forwarding packets. This request is routed through the new BS. The arrows represent the logical endpoints of a message, not the path that the message takes from source to destination. Downward vertical handoffs are initiated when several beacons are heard from a lower overlay's NI. The MH determines that the mobile is now within range of the lower overlay's NI and switches to the lower overlay. The handoff starts when the lower overlay becomes reachable or unreachable, and ends when the first data packet forwarded from the new overlay network arrives at the MH. As previously mentioned, our system only depends on the presence or absence of packets to make vertical handoff decisions.

2.3 Techniques for Low-Latency Vertical Handoffs

One of the goals in our handoff system is to support interactive multimedia communication across multiple network interfaces, and for these applications, a latency measured in seconds is unacceptable. Even for non-real time applications such as non-interactive file transfers and WWW browsing, a latency of several seconds will lead to a loss of multiple data segments. Previous work has also shown that packet losses during handoff has detrimental effects on reliable transport protocols such as TCP [Cac95]. With this in mind, we examined several enhancements to the base strategy that allow us to reduce handoff latency.

2.3.1 Hints for Enabling Enhancements

The schemes described in this section are used in situations where the application indicates that a low handoff latency (less than 300-500ms) is important, such as real time interactive voice or video. Even when an application indicates that low-latency handoff is important, these enhancements are not used continuously, because of bandwidth/power overheads. They are used only when the mobile is in a situation where it may hand off soon. Note that this is not the same as determining that a mobile must hand off immediately (i.e., the mobile is now disconnected). Alternative hints can be used to predict that a handoff is likely. These include

- *User input*: The user can instruct the MH to be more aggressive about handoff by using these enhancements. When the user is likely to leave the building, she can put the MH in a mode that uses these enhancements. The user can take the MH out of this state when not moving.
- *Received signal strength*: Although signal strength indicators, when present, may not be a good indicator of imminent handoff, they do well at indicating the distance between a MH and BS. When a MH notices that the signal strength is gradually decreasing it can assume that the user is moving away from a BS, and when the signal strength is increasing a MH can assume that the user is moving toward a BS. When the best BS that a MH can hear has a low signal strength that has been decreasing, a MH can assume that a vertical handoff may be likely and start using some of these enhancements.
- *Geographic hints*: We can use traces to predict which cells are the gateways to a new overlay network. Although the overlapping nature of wireless overlays means that a user can be potentially connected to multiple networks at once, the transitions between networks are a function of the building geography. A vertical handoff is only possible from certain places in the building, and only certain cells cover these locations (e.g. only one in-building RF cell is likely to cover the exit of an office building). The BSs covering these cells could add information in their beacon packets indicating that this cell is near the exit to a building, and that a vertical handoff to a wide-area network is likely.
- *Handoff Frequency*: The MH can also track the frequency of handoffs and use these enhancements when more handoffs are occurring, indicating that movement out of this overlay's coverage is more likely. This approach has been suggested for switching between high-tier and low-tier PCS systems [Tek91].
- *Missing a single beacon*: We mentioned in Section 2.2 that the MH waits for multiple beacon packets before determining that an overlay is (un)reachable and switching to a new overlay. The MH could turn on some of these optimizations after missing a single beacon packet, as an attempt to verify that an overlay is (un)reachable.

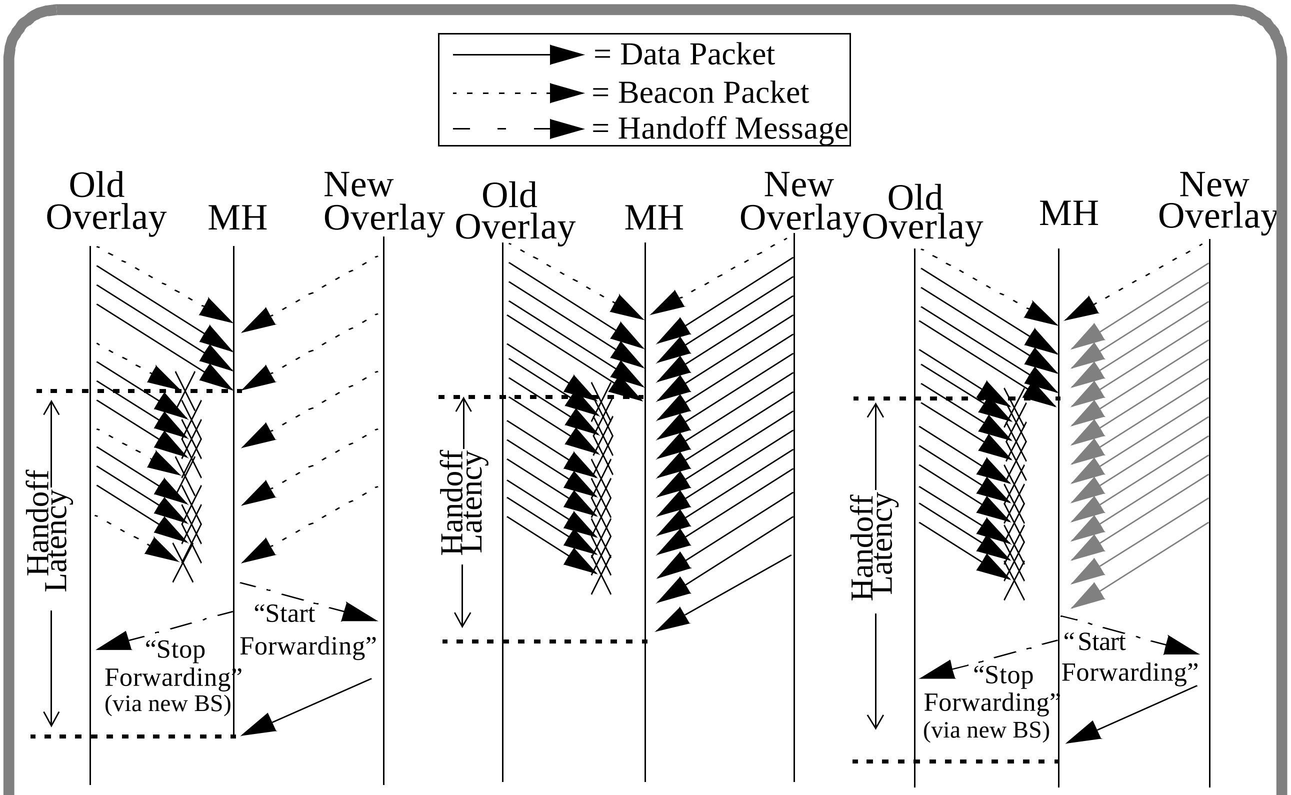


FIGURE 6. (a) Breakdown of Fast Beaconing Handoff

FIGURE 6. (b) Breakdown of Packet Doublecasting Handoff

FIGURE 6. (c) Breakdown of Header Doublecasting Handoff

2.3.2 Enhancements

We can make the following enhancements to reduce handoff latency. All of these enhancements have some additional cost in terms of power or overhead bandwidth.

- **Fast Beaconing (Figure 6(a))**: The MH can selectively instruct a subset of the BSs that are listening to its multicast group to transmit beacon packets at a higher frequency than once per second. The MH still waits for several beacons to be lost before initiating a handoff, but the beacons are transmitted more quickly and the handoff latency is reduced. The breakdown of a handoff is described in Figure 6(a). The handoff proceeds exactly as in Figure 5(b) — the beacon packets are simply received more quickly.
- **Packet Doublecasting (Figure 6(b))**: The MH can place into forwarding mode a subset of the BSs that are listening to the multicast group for the MH. This means that multiple copies of the packet will be transmitted from multiple BSs to the MH. In our scheme, two BSs are placed in forwarding mode simultaneously; the current BS and a BS of the next higher overlay. Duplicate packets are filtered out at the network layer at the MH by keeping a small cache of received IP packets and filtering out received packets whose IP ids are already in the cache. Although not strictly needed, this prevents unnecessary congestion control mechanisms from being invoked at the transport layer. The network layer at the MH also keeps track

of packets that have been received by each interface. When more than a threshold number of consecutive packets are received on a new interface with none received on the old interface, the MH decides that the old overlay is unreachable and initiates a handoff to the new interface. A breakdown of the handoff is shown in Figure 7. Two copies of each packet are sent to the MH, one from each BS. After 10 packets are missed from the old overlay, the mobile switches to the new overlay. The packets kept in the network-level cache on the MH are forwarded to higher layers. In cases where no data is currently being sent to the MH, beacons are used to trigger a handoff. By utilizing diversity that arises from multiple network interfaces, this approach does at the network layer what the IS-95 CDMA Cellular phone standard [Lee94] and the ARDIS wide-area data system [Ardis] do at the physical layer. In IS-95, multiple BSs send duplicate copies of the same data using the same CDMA codes. The MH's receiver is already equipped to handle multiple time-shifted copies of the same waveform, and a MH moves into the cell of the new BS seamlessly. In ARDIS, multiple BSs transmit the same data at the physical layer to achieve better in-building penetration.

- **Header Doublecasting(Figure 6(c))**: This approach takes advantage of the fact that in the Packet Doublecasting approach, duplicate packets on the upper interface are used only as an indicator of handoff. Therefore, full packets do not have to be sent until the actual handoff occurs. In this approach the MH places a BS into a mode where it continues to buffer packets destined for the mobile host. However, the BS also forwards a packet containing the IP header of the buffered packet to the MH. The network layer at the MH keeps track of which packets or packet headers has been received by the mobile. The MH switches to the new BS when more than a threshold number of headers have been received via new BS while no packets have been received via the old BS. The new BS forwards the packets just as in the Basic System. This approach has an advantage over Packet Doublecasting in that less data is sent on the upper overlay.

Both doublecasting approaches have an advantage over the beaconing systems in that they use extra resources only when the MH is actively receiving data. When the user is not receiving data, no extra bandwidth is used. Additionally, beacons sent from the base station affect all mobile devices in the wireless cell, and if beacons are sent at very high frequencies, media access affects (such as exponential backoff during link activity) may dramatically reduce the effective bandwidth of mobile hosts in the same cell. Another advantage of the packet doublecasting approaches is that the packets that trigger a handoff are not redundant; they are consumed by actual applications. If fast beaconing were used, then beacons (useless application-level data) would be competing with application-level data for network resources at all times.

A disadvantage of the doublecasting approaches is that both overlays must be able to support the same network load. Packet doublecasting across a high-bandwidth and low-bandwidth network will not work. Another advantage of the beaconing systems is that multiple users in a cell can use the same beacon packets (rather than separate data packets) to make handoff decisions.

2.4 Differences between Vertical Handoff system and IETF Mobile IP

Although our system is based on Mobile IP, there are currently some differences between our implementation and Mobile IP:

- Our beacon message formats and communications between the MH and BS do not exactly match those used in Mobile IP. This is mainly due to the time when the

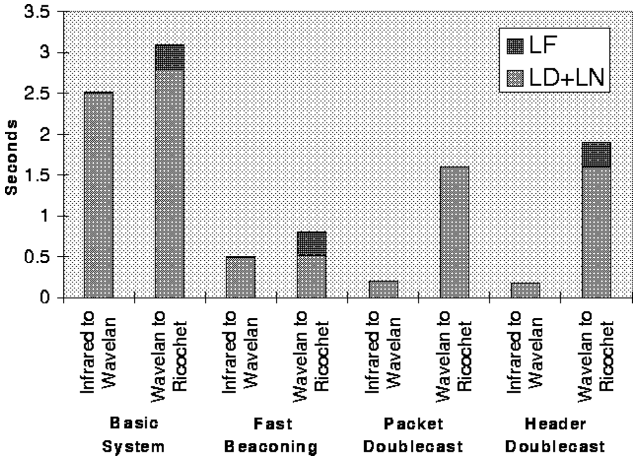


FIGURE 7. Comparison of Handoff Latency for Basic System and Enhancements

Horizontal handoff system was implemented (in parallel to the formalization of the Mobile IP specification).

- Our system assumes trust between MHs, FAs, and HAs. As a result, our implementation does not implement the MH-HA and MH-FA authenticators described in [Per96].
- The care-of-address in our system is a multicast address, so there is no explicit communication between the FA and the HA.

The above differences could easily be resolved by making our implementation conform to the Mobile IP one (i.e. changing message formats, adding functionality, etc.). There are some differences, however, that cannot be resolved easily:

- Because of the optimizations we use for low-latency vertical handoffs, there are extra communication types between MHs and FAs (i.e. forward packet headers, beacon faster, etc.) which cannot be expressed only using Mobile IP messages.
- Although the policy that the MH uses for initiating handoffs is not explicitly specifcied in Mobile IP, our policy for Vertical Handoffs is significantly different than the policy that most Horizontal Handoff systems use. We describe these differences in more detail in [Ste97].

2.5 Overlay Networking Summary

Overlay networking allows a mobile device to detect new networks and to switch to the best available network as the user moves around. Combined with the fast handoff optimizations, this provides fast, transparent access to the best network.

Figure 7 summarizes the performance of the basic handoff system and each of the enhancements for each of the upward vertical handoffs. We have learned the following things about the enhancements proposed to reduce handoff latency:

- Fast beaconing results in a decrease in latency proportional to the bandwidth overhead. This approach consumes bandwidth whether or not data is being sent to the mobile device.

- Packet doublecasting results in a loss-free zero latency handoff, but at a prohibitive cost.
- Header doublecasting results in a latency similar to the packet doublecasting scheme, but with a dramatic decrease in overhead.
- For handoffs between in-building and wide-area overlays, doublecasting approaches have limited effect due to the latency-bound nature of the wide-area network we used.

For the network interfaces in our overlay network structure, header doublecasting performs the best for transitions between in-room and in-building networks, and beaconing works best for transitions between in-building and wide-area networks.

By providing seamless coverage to mobile clients as they roam in an Overlay Network structure consisting of heterogeneous networks, the overlay networking subsystem meets the previously described principles of **Scalability**, **Transparent Access**, and support for **Heterogeneous Networks**. By using optimizations to keep handoff latency low, we also meet the principles of **Multimedia** support and high **Performance**.

3 Reliable Data Transport Performance

A mobile networking environment presents several challenges to the problem of efficient reliable data transport. In this section, we discuss the challenges posed by the presence of different types of wireless links, and then present our solutions to them. In this discussion, data transport refers to the transfer of bytes *end-to-end*, i.e. from a source host to a destination host. The performance of data transport can be quantified using several metrics such as throughput, goodput, latency and fairness. Given the emphasis on end-to-end transfer of data (which is, after all, what really matters to an end user), it is clear that any link (or subnetwork) in path from source to destination has the potential to influence performance. In practice, performance is determined primarily by the weakest link in the chain. As we elaborate in the remainder of this section, a mobile subnet is often such a weakest link in the chain.

The de-facto standard protocol for reliable data transport in the Internet is TCP[rfc1123,Ste94]. TCP provides a reliable byte-stream abstraction to the higher layers, delivering bytes in-order to the application. It achieves reliability by using *cumulative acknowledgements (acks)*, which are sent by the recveiver to the sender for all received data on a regular basis. These acks are used by the sender to determine the successful delivery or loss of data. In addition to this *error control*, TCP uses these regular acknowledgements to pace out new data at a consistent rate, a procedure called *ack clocking* [Jacobson88]. In the absence of any a priori knowledge of the state of the network, TCP uses the rate at which acks arrive to deduce what rates can be sustained on that connection. Therefore any irregularity in the acknowledgement stream gets manifested in the data stream in the immediate future. To avoid overwhelming the network, TCP performs congestion control by gradually opening its transmission window through the *slow start* and *linear growth* phases, and cutting back its window by half or more when data loss happens. An implicit assumption here is that data loss is caused by congestion, a valid assumption in traditional wired networks.

Many of the assumptions that TCP makes break down in mobile subnets that have different kinds of wireless links. First, wireless links tend to be error-prone, resulting in bit-errors. This is the result of interfering noise due to other wireless transmissions and/or self-interference (fades) due to reflections from physical objects in the environment. Wireless losses make TCP's assumption, that data loss is always due to congestion, invalid, and often result in degraded end-to-end performance. Therefore, it is desirable to shield TCP from such losses, or make it aware of different types of network losses, in order to obtain good performance in networks with wireless links. Also, since the mobile subnet with its wireless links tends to be located at the periphery of the internetwork, it is often desirable to employ a local solution that does not impact the rest of the network. These considerations motivate our work on a transport-aware reliable link-layer protocol, called the *snoop protocol,* described in Section 3.1.

A second problem arises due to network asymmetry. Due to technological and economic considerations, it is typically much easier to have a high bandwidth and more reliable link going from a central base station to mobile nodes (the *forward* direction) than in the opposite (*reverse*) direction. In terms of *bandwidth*, the extent of asymmetry can range from 1:10 ro 1:1000. Examples of such networks include broadband satellite or LMDS-based forward channels coupled with a dialup reverse channel, say using the cellular phone or packet radio network. Another form of asymmetry, called *latency asymmetry*, arises due to the nature of the media-access protocols in several packet radio networks, and results in large and variable delays. In both these cases, the process of transferring data in one direction is significantly influenced by the traffic in the opposite direction. The net result of each of these forms of asymmetry in the context of TCP is that ack clocking gets disrupted, thereby resulting in degraded performance. Our approach to mitigating this problem encompasses both techniques that operate end-to-end and those that operate just locally in the wire-

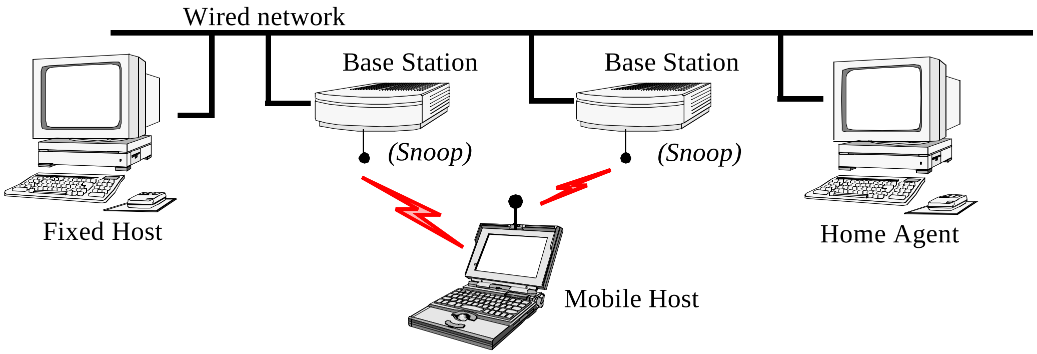


Figure 1. Network topology of the single-hop cellular wireless network based on the WaveLAN. The snoop agent runs at the base stations in the network.

less subnet. As we discuss in Section 3.2, it is possible to use either approach to achieve similar performance benefits.

The third problem that we address arises due to the characteristics of the media-access control (MAC) protocol used in wireless networks, especially wireless LANs. The MAC protocol used in such networks are derived from the corresponding protocols used in wired networks. However, wireless networks are different from wired networks in certain important ways, which could significantly impact data transport performance. For example, it is difficult to do collision detection in wireless networks. Also, the quality of connectivity could be significantly different for each receiver at the same point in time. Our approach to addressing these problems is XXX Giao, please complete

XXX incorporate this into previous paragraph Finally, in the context of improving network performance, we also discuss the issues of link sharing in wireless local area networks. The goal of link sharing is to provide a controlled bandwidth allocation for mobile hosts and their applications, and thus improving the scalability of the network. TCP performance would benefit from such a mechanism because of the stable link usage in the wireless hop provided by link sharing.

In summary, our approach to improving data transport performance in mobile, wireless networks has been identifying the specific characteristics of such networks (e.g., errors vs. congestion, asymmetry, unfairness, etc.) that impact TCP performance and then incorporating end-to-end as well as local mechanisms to either make TCP adapt better to these characteristics or to hide them from TCP. By focusing on TCP, we are able to make our solutions applicable to the vast collection of applications on the Internet that use TCP, and help in integrating wireless technologies into the global Internet.

3.1 The Snoop Protocol

Over the past several years, TCP has been tuned to perform well in traditional networks made up of links with very low bit-error rates. Networks with higher bit-error rates, such as those with wireless links and mobile hosts, violate many of the assumptions made by TCP, causing degraded end-to-end performance. In particular, packet losses are assumed by TCP to be due to congestion, rather than because of errors or user mobility, and this causes the invocation of congestion control mechanisms (e.g., slow start [Jac88]) that decrease the sender’s congestion window and reduce throughput.

The snoop protocol is a localized protocol that modifies the networking software only at the base station and at the mobile host connected over the wireless link. In particular, no modifications are required to TCP implementations elsewhere in the Internet. The protocol works by deploying an agent called the snoop agent at the base station. The agent mainly performs the functions of loss detection and loss recovery via local retransmissions. The network topology and snoop agent location are shown in Figure XXX.

For transfer of data from a fixed host *to* a mobile host, we make modifications only at the base station. These modifications include caching unacknowledged TCP segments and performing local retransmissions based on a few policies that depend on TCP acknowledgments from the mobile host and timeouts of locally-maintained timers. By using duplicate acknowledgments to quickly identify packet loss, performing local retransmissions as soon as data loss is detected, and *suppressing* these duplicate acknowledgments from the TCP sender, the snoop agent shields the sender from the vagaries of the wireless link. In particular, transient situations of very low communication quality and temporary disconnectivity are completely hidden from the sender.

For transfer of data *from* a mobile host to a fixed host, the snoop agent detects missing packets at the base station by snooping on packets arriving from the mobile host and identifying holes in the transmission. Then, when it sees duplicate acknowledgments arriving from the receiver that signifies a loss of this packet, it sets a bit in its TCP header and forwards it to the mobile sender. The sender uses this *Explicit Loss Notification (ELN)* bit to identify that the loss was unrelated to congestion, and retransmits the packet without taking any congestion-control actions. This requires modifications to both the base station and mobile hosts.

A key design goal in this work is the use of only *soft state* in the protocol. The state maintained by the snoop protocol is *soft*, implying that its loss or corruption does not impact the correct functioning of the protocol. We achieve this by taking advantage of the fact that TCP acknowledgments are cumulative, which makes it easy to regenerate and periodically refresh any out-of-date state. The snoop protocol should be viewed strictly as a performance improvement, which greatly improves the performance of TCP over single-hop wireless networks. If the state associated with the protocol is lost, it can be rebuilt rather easily upon the arrival of the next packet and acknowledgment.

The mechanisms described above together improve the performance of connections in both directions, without sacrificing any of the end-to-end semantics of TCP or modifying host TCP code in the fixed network. This combination of greatly-improved end-to-end performance, preservation of end-to-end semantics, and purely local recovery techniques is the main contribution of this work.

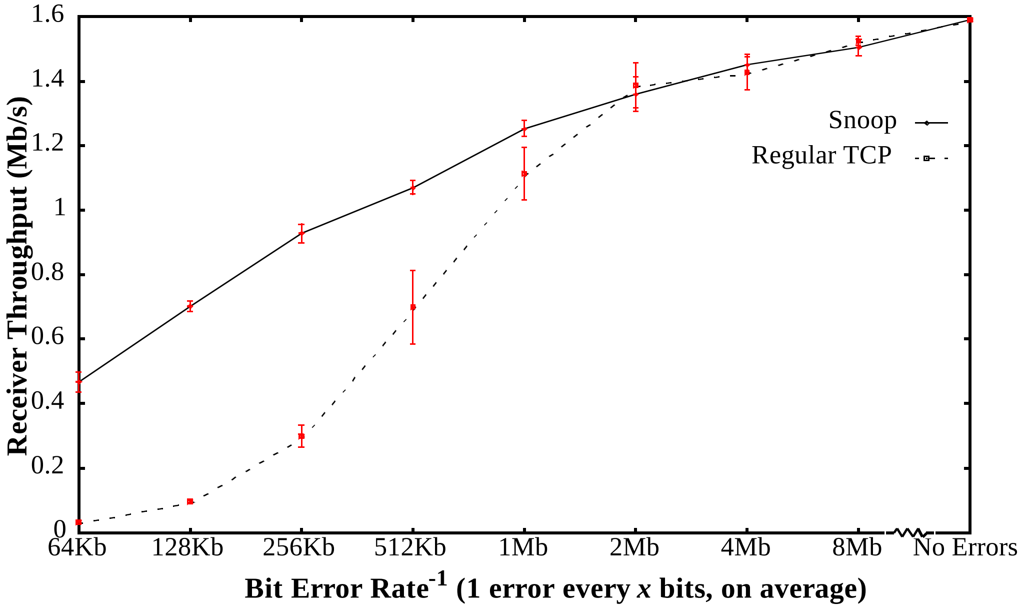


Figure 2. Throughput received by the mobile host at different bit-error rates ($\log_2$ scale). The vertical error bars show the standard deviations of the receiver throughput.

We have implemented a prototype version of the snoop protocol on a wireless testbed consisting of IBM ThinkPad laptops and Pentium PC basestations communicating over a 2 Mbps AT&T Wavelan. Our experiments show that it is significantly more robust at dealing with unreliable wireless links as compared to normal TCP; we have achieved speedups of between 2 and 20 compared to regular TCP in our experiments with the protocol over a WaveLAN link across a range of simulated bit-error rates, as shown in Figure XXX. We have also performed an extensive performance evaluation of the snoop protocol by implementing several other schemes (including other flavors of link-layer protocols, end-to-end protocols and split-connection protocols) and comparing theirelative performance [BPSK96]. Our results demonstrate the significant benefits and advantages of such transport-aware link layer schemes, in addition to demonstrating the effectiveness of ELN and selective acknowledgments in handling non-congestion-related packet losses.

We have also integrated the multicast-based low-latency handoff scheme described in Section 2 with the snoop protocol for improved TCP performance in the presence of both wireless bit-errors and user mobility between cells. The idea is to take advantage of the multicast and intelligent buffering at the base stations (used to achieve low handoff latencies) in order to perform the mirroring of the snoop agent's soft state as well.Further details

3.2 Asymmetric Networks

As mentioned in the previous section, TCP relies on the ack clocking mechanism to inject new data into the network at an even pace. In effect, TCP depends on the smooth flow of packets (acks) in the opposite direction in order to sustain good data transport performance in a certain direction. While this dependence in fact leads to robust congestion control in most wired networking scenarios, it can result in degraded performance in networks that exhibit asymmetry, such as many wireless networks.

Given our focus on TCP, we have the following characterization of network asymmetry: *a network is said to exhibit asymmetric characteristics with respect to TCP, if the data transport performance in one direction is affected significantly by the traffic and network charcteristics in the* *opposite direction*. Note that the traffic in the opposite (reverse) direction could just be the TCP acks for data in the forward direction.

This general definition leads to several types of asymmetry that we can classify:

- *Bandwidth:* The bandwidth in the forward direction (towards the end user) is much larger (10 to 1000 times larger) than that in the reverse direction (away from the user). Networks based on Direct Broadcast Satellite (DBS) systems or wireless cable modem technology exhibit such asymmetry.
- *Latency:* In certain wireless networks, such as Cellular Digital Packet Data (CDPD), the underlying media access control (MAC) protocol often results in a significantly larger one-way latency from a base station to a mobile station than in the reverse direction. In other networks, such as the Metricom Ricochet multi-hop wireless network, there is a significant delay when a node switches from sending to receiving mode or vice versa, a phenomenon that is exacerbated in the poresence of bi-directional traffic (e.g., caused by TCP acks).This often results in significant variations in round-trip times and makes it hard for TCP to adapt to the characteristics of the network.
- *Bit-error rates*: The incidence of packet errors could be much greater in the upstream direction than in the downstream direction. This could be inherent in the network technology (e.g., a 2-way cable modem system) or the result of distinct upstream and downstream technologies (e.g., a DBS downlink with a packet radio uplink).

We observe that the reason for the development and deployment of asymmetric network access technologies is in part due to technological and economic considerations that make it easier to provide a fast and reliable channel out from a central base station than in the opposite direction, and in part because of the asymmetric communications requirements of today's popular applications such as Web access.

We identify three specific problems that network asymmetry results in as far as TCP performance is concerned. We discuss both end-to-end techniques (i.e., enhancements to TCP) and local techniques to alleviating these problems, which are *generally applicable* across a variety of networks that exhibit asymmetric characteristics.

First, the ack stream corresponding to a data transfer could either lead to congestion over the bandwidth-contrained reverse channel thereby disrupting the flow of data packets indirectly (due to the breakdown of ack clocking) or interfere directly with the flow of data packets (as in a packet radio network). Our basic solution to this problem is to decrease the frequency of acks transmitted by the TCP receiver. Our end-to-end scheme, called *ack congestion control*, allows the receiver to adaptively reduce the frequency of acks, while ensuring that the sender is not starved of acks. This requires the TCP sender to tell the receiver how large its window is. Our local scheme, called *ack filtering*, operates at the router to the constrained reverse channel. It exploits the cumulative nature of TCP acks to filter out redundant acks from the queue when a new ack for the same connection arrives at the input to the queue.

However, the decreased frequency of acks leads to problems at the TCP sender. Since each new ack could cause the sender's window to slide forward by several segments, the sender could emit a burst of several packets back-to-back, and induce congestion in routers downstream. In addition, since the TCP sender increments its congestion window based on the *number* of acks and not on how many bytes are acknowledged with each ack, window growth could now be much slower. Our end-to-end mechanism for combating these problems, called *sender adaptation*, breaks up a potentially large burst into several smaller bursts and spaces apart these smaller bursts according to the

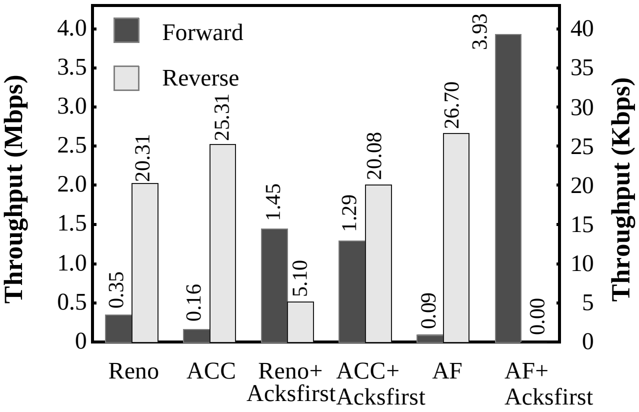


FIGURE 8. Throughput of a 3 MB forward transfer and a 100 KB reverse measured in a real asymmetric network (10 Mbps forward and 28.8 Kbps reverse bandwidth). The reverse transfer is initiated 2-3 seconds after the forward transfer. Only the time during which both transfers are active is considered for computing throughput. Note that the reverse throughput is 0 for the "AF+Acksfirst" case.

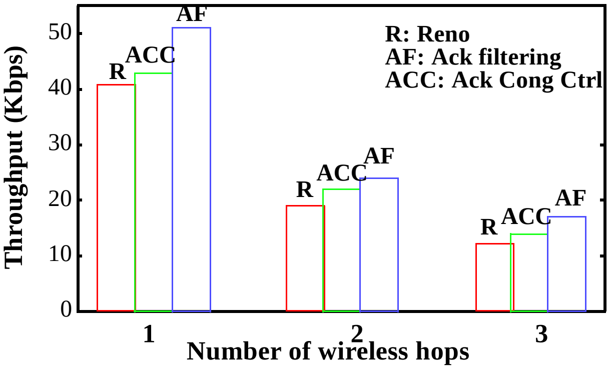


FIGURE 8. TCP throughputs from simulations of Reno, ACC and AF, as a function of the number of wireless hops. Parameters derived from the Ricochet packet radio network were used for modelling the network.

effective data transfer rate of the connection. Our local solution, called *ack reconstruction*, hides the infrequent ack stream from the TCP sender by deploying an agent at the other end of the constrained reverse channel. Once the acks have traversed the constrained network, the ack reconstructor attempts to reconstruct the original smooth ack stream by inserting new acks to fill in gaps and by spacing apart acks that have been bunched-up. The end result is that an unmodified TCP sender can continue to rely on standard ack clocking to sustain the data transfer at a consistent rate.

The final problem arises specifically in an asymmetric-bandwidth situation where the acks of the forward-direction data transfer have to share the constrained reverse channel with a reverse-direction data transfer (simultaneous bi-directional data transfers). With FIFO queuing, the large data packets of the reverse transfer could block the transmission of acks for long periods of time, thereby starving the sender of the forward transfer. Our local solution to this problem, called *acks-first scheduling*, involves giving acks a strictly higher priority than data packets. The rationale is that the small acks packets when sent infrequently would have little impact on the performance of the reverse data transfer, but their timely transmission is critical to sustaining good performance for the forward data transfer. Since the resolution of this problem requires control over the scheduling of data and acks at the reverse bottleneck link, we do not have an end-to-end version of the solution.

We have done a detailed evaluation of these solution techniques both in a the *ns* simulator environment as well as in our network testbed. These results are discussed in detail in [BPK97] and [BPK98]. We present here sample results for both the asymmetric-bandwidth and asymmetric-latency configurations. Figure 8 shows the performance of various schemes when there is two-way traffic in a network with a high-degree of bandwidth asymmetry. These were measurements done using a real implementation of the schemes in our testbed. We make two important observations. First, acks-first scheduling helps the forward-direction transfer significantly by avoiding the long waits that its acks would have had to encounter at the reverse bottleneck had FIFO scheduling been used. Second, ack congestion control (ACC) helps ensure that the acks of the forward transfer do not result in starvation of data packets of the reverse transfers. It ensures this by decreasing the frequency of acks.

Figure 8 shows the performance of various schemes obtained via a simulation of the Ricochet packet radio network. The important observation is that ack filtering (AF) and to a lesser extent ack congestion control (ACC) result in improved performance. The reason is that the decrease frequency of acks in these cases decreases MAC contention and hence latency variability.

3.3 Link Sharing for Wireless LANs

In this section, we present the designs of a link sharing mechanism in a wireless LAN environment to improve the scalability of such a network. Wireless LANs (WLAN) are placed in the bottom layer of the wireless overlay network. They generally provide shared access to a single channel (hundreds of Kbps to a few Mbps) using CSMA based protocols. Examples include the Proxim RangeLAN and AT&T WaveLAN[DN95]. The bandwidth provided by a WLAN, while better than most other wireless network technologies, is still limited. An effective link sharing mechanism to provide stable network performance to applications and mobile hosts is important and necessary for a wireless network to accommodate more users and demanding applications. Furthermore, it is also desirable for administrative reasons to be able to control the link resource allocation. For example, while providing access to roaming mobile hosts, the owner of a WLAN may want to ensure that the local users are given larger slices of the bandwidth.

Essentially, there are two most important goals for an effective WLAN link sharing mechanism.

1. *Bandwidth Sharing*: A link sharing mechanism's primary responsibility is to allocate bandwidth among mobile hosts and to enforce the allocation.
2. *Surplus Bandwidth Distribution*: If a mobile host is not using its allocated bandwidth, the surplus bandwidth should be shared by other mobiles hosts in a controlled fashion.

The general issues of link sharing are well discussed in the modeling of Class Based Queueing (CBQ) in [FJ95]. CBQ provides effective hierarchical link-sharing on a point-to-point link, and it has being implemented in the Trans-Atlantic FAT pipes, among others. However, the multi-access nature of WLANs requires a distributed link sharing mechanism, which is the subject of the following discussions. The goal of this study is, quite simply, to extend CBQ to the WLANs.

We base our discussions on WaveLAN, which is a 2 Mbps WLAN. We also assume that all mobile hosts and the base station follows the link sharing protocol. Furthermore, there is no direct communication among mobile hosts in a WaveLAN.

3.4 Bandwidth Sharing

In this subsection we describe the partitioning of the bandwidth among mobile hosts and the enforcement of such bandwidth sharing guidelines.

Figure 9 shows a hierarchical link sharing structure used for partitioning the bandwidth among agencies and further among sub-classes of each agency in [SJ95]. We use a similar link sharing structure, as shown in Figure 10, except the concept of an agency is substituted by a*node*. A node is either a mobile host or a basestation in a WaveLAN. Because both up-stream and down-stream traffics share the single channel, it is necessary to cover both traffic directions in partitioning bandwidth.

The enforcement of such a hierarchical bandwidth allocation guideline is performed at two levels. Firstly, a node need to ensure that its internal link sharing guideline is followed. Since this

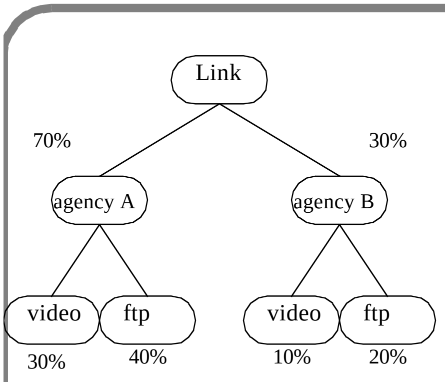


FIGURE 9. Link Sharing Structure for CBQ

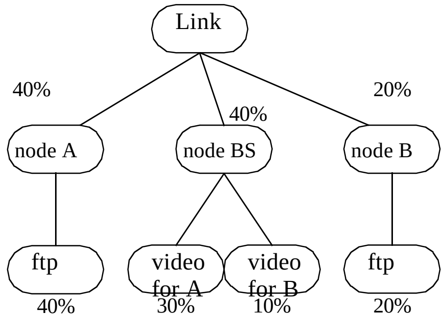


FIGURE 10. Link Sharing Structure for WLAN

requirement is no different from the CBQ goal, the same scheduling algorithms discussed for CBQ can be used.

Secondly, each node needs to get its allocated bandwidth. This is done with a very simple method. The basic idea is that if bandwidth is not over-allocated among nodes, and that each node restrains itself at the link level from using more than its allocated bandwidth, everyone would receive their promised bandwidth share. There is no explicit coordination among nodes for the link usage, and no state exchange is necessary besides the initial network cell admission and subsequent bandwidth allocation process.

Figure 11 shows a simulation result. In this simulation, there are 9 nodes using a WaveLAN link. One node is sending a CBR stream at 10% of the total link bandwidth, and the packet size is 190 byte. All other 8 nodes send random sized packet ranging from 200 byte to 1000 byte. 3 of these 8 nodes are allocated 20% of the total link bandwidth each, and the rest 2% each. All these 8 nodes have sufficient demand if the bandwidth is available. The line at the top is the total link usage. The *x*-axis shows time in seconds. The *y*-axis shows the average bandwidth used by each node over 1 second intervals, as a percentage of the WaveLAN link bandwidth.

FIGURE 11. Bandwidth Sharing in WaveLAN

As shown in the figure, the enforcement of the bandwidth sharing works very well. Each node receives about its allocated bandwidth, and the total link usage is close to 80%.

3.4.1 Surplus Bandwidth Distribution

The issue of surplus bandwidth distribution in WaveLAN is one of state exchange. For example, when a node is not using its share of the bandwidth, some other nodes need to get this information to take advantage of the surplus link resource. Firstly, we need to define the concept of surplus bandwidth. In a wireless link, the number of packets injected into the link is often different from the number of packets coming out of it, due to the wireless errors. For the accounting purpose in this link sharing mechanism, the bandwidth used in terms of injected packets is used to measure the link usage. Surplus bandwidth, therefore, is the difference between the attempted bandwidth usage and the bandwidth allocation of a sub-class or a node. In this subsection, we describe the approach to state exchange for each of the three types of surplus bandwidth distribution: within a node, between the up-stream and down-stream of a mobile host, and among the nodes.

State exchange within a node is straight forward. When one particular sub-allocation for a node is not being used, other sub-allocation connections can take advantage of it, given sufficient demand. The algorithms discussed in [FJ95] for CBQ can be directly applied.

State exchange between the up-stream and down-stream of a mobile host is a special case due to the shared channel property of WaveLAN. The exchange of information can be largely implicit. Because the two nodes involved, the base station and a mobile host, are both senders and receivers for each other. In general, each can monitor the other's link usage to be able to understand the other's states and therefore surplus bandwidth. To account for packet losses due to error, some regular updates should be used to adjust the state exchange between the two. These updates can be simple and piggybacked on regular data packets to reduce overhead.

State exchange among nodes, however, need to be explicitly conducted. When a mobile host is not using its combined allocation for up-stream and down-stream link bandwidth, the surplus link resource should be distributed among other nodes according to certain policy. In this case, the base station is at the position of collecting up-to-date link usage information, which is already collected for the purposed stated in the previous paragraph. It is up to the base station to inform other nodes to adjust their link usage. The central role of the base station is due to two reasons:

1. Administratively, the base station is naturally the one to enforce the WLAN usage policy.
2. Because most of the traffic in a WLAN is expected to be heavily biased towards the downstream traffic, it is really most likely an internal state exchange at the base station for it to distribute the surplus bandwidth among the other down-stream traffics of other mobile hosts. This also means that the explicit state exchange from base station to other mobile hosts is not necessarily frequent or expensive.

3.5 Summary

Taken together, our extensions provide a faster and much more robust version of TCP. These modifications are critical for wireless networks, but are more widely applicable. Thus these extentions enable improved performance in heterogeneous networks comprising both wireless and wireline components.

4 The Scalable Proxy and Coordination Bus

The proxy's main tasks are to shield clients from the effects of slow networks and to tailor Internet content to the needs of each different client, allowing meaningful content presentation across all clients. As described in detail elsewhere [FGBA96], we have found that*datatype-specific lossy compression*, which we call *distillation*, is an effective adaptation mechanism. Distillation provides well-defined operations over semantically typed data. For example, distillation of an image consists of selectively discarding color information, high-frequency components, or pixel resolution. Distillation of video can additionally include frame-rate reduction. Less obviously, distillation of formatted text requires discarding some formatting information but preserving the actual prose. In all cases, the goal is to preserve information that has the highest semantic value, if necessary changing its format to enable optimal presentation on the target device.

The user can always explicitly ask for a higher-quality representation of degraded content later, if she decides that the data is valuable enough to be worth the additional latency; for instance, zooming in on a graphic or video frame, or rendering a particular page containing PostScript text and figures without having to render the preceding pages. We define*refinement* as the process of fetching some part (possibly all) of a source object at increased quality, possibly the original representation. As with distillation, the refinement technique is a function of the semantic type, and the implementation of the technique requires intimate knowledge of the encoding. For example, "zooming in" is a useful operation for all images regardless of encoding, but encoding-specific tools are necessary to extract and magnify the desired subregion.

Providing distillation and refinement at the proxy confers several architectural advantages. First, it allows a wide range of mobile clients using networks with widely varying characteristics to interoperate transparently with legacy servers, since all adaptation is performed at the proxy, which looks like a powerful, well-connected client from the server's point of view. Even if we could modify servers, however, performing adaptation at the proxy confers an important separation of concerns: the servers focus on content provision, while the proxy focuses on individualized content presentation. This separation has the technical benefits of improved maintainability and amortization of proxy resources, and the economic benefit of orthogonalizing content provision and quality of service provision, which decouples both administration and billing (distillation and refinement represent a value-added service to users with slow network connections or impoverished clients).

In the following subsections, we describe the internal architectural organization of a scalable proxy designed to provide individually-customizable service to each of thousands of users, based on each user's personal preferences, client characteristics, and network characteristics. The network characteristics can be explicitly supplied (if a "stock" network stack is used) or deduced by the Sessions layer described in XXXX (if the Dædalus network stack is used). Although the proxy functionality is central to our overall architecture, the narrow interfaces used by the proxy (in this case, the HTTP proxy interface) enables it to operate as a standalone component without the rest of the architecture as well, and in fact we have already deployed Web proxy service to the dialup user communities at UC Berkeley and UC Davis using this platform.

4.1 The Coordination Bus

The coordination bus [FJM+95] provides a virtual bus with discrete, symbolically named channels, layered on top of a multicast network. A coordination-bus-aware program can broadcast information on one or more CB channels as well as hear the broadcasts of other CB-aware programs by listening on the appropriate channel(s). CB channels can be used for communication between different systems or between components of a single distributed system. Our scalable proxy is an example of a non-trivial distributed system that uses the CB in both of these ways: various components of the system use the CB to exchange information used to achieve fault tolerance and automatic load balancing. In addition, the "front end" (the part of the system visible to the outside world) can be controlled by sending messages on the CB.

The scalable proxy defines the following channels:

GM_PTMBEACON: As described in Section 4.2.3, the PTM sends out periodic beacons so that workers and the front end can find it. If the front end detects that the beacons have gone away, it assumes the PTM has died, and attempts to launch a new one.

GM_NETSTATE: The front end can be advised of automatically-detected network changes via this channel. A network state notification specifies which user(s) the change applies to and the new values of the changed network characteristic(s), such as bandwidth and latency.

4.2 The Proxy Architecture

In this section we give a high-level description of the internal architecture of the scalable proxy for distillation and refinement. A detailed description, including performance information and fault-injection experiments, may be found in [FGC+97]. The proxy is designed to run on a well-connected network of commodity workstations (NOW). Although the core software is considerably more general, we focus our attention here on the particular mechanisms that allow the scalable proxy to work as an HTTP proxy [Luo94] for the World Wide Web. The design goals of the proxy are as follows:

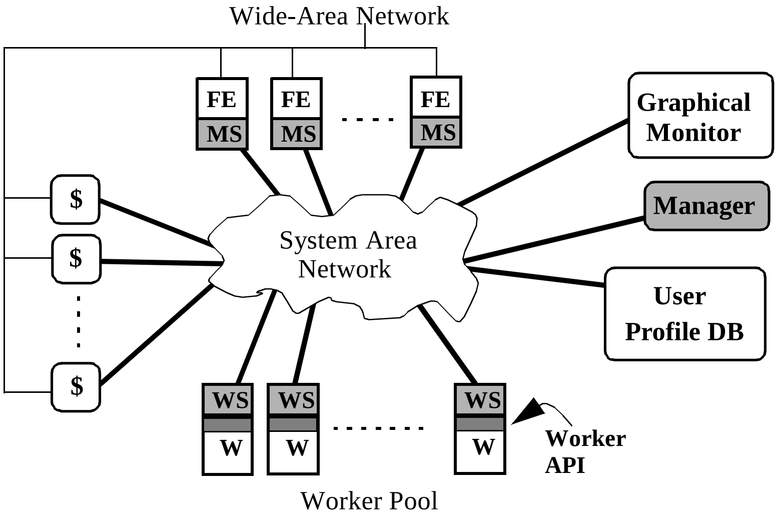
- **High performance**: the latency of serving an individual user should be minimized. Ideally, the overhead of using the proxy should be barely noticeable even when a high-speed connection is used. This supports Principle 8 (**Performance**) from the Introduction.
- **Scalability**: as the number of users increases, the per-user performance can be held approximately constant by adding nodes to the NOW. This supports our goal of serving thousands and eventually millions of users (Principle 2 from the Introduction).
- **High availability**: the ability to transparently mask the transient failure of any individual component, so that the service as a whole is continuously available. Meeting this design goal is a requirement of Principle 3 (**High Availability**) from the Introduction.
- **Mass customization:** A persistent *preference profile* is maintained for each user of the proxy. The profile is automatically used to tailor the proxy's behavior on every request from that user.

We now describe the components of the proxy, which are shown in Figure 12. Note that all components except the Proxy-Transcoder Manager and User Preferences Database are replicated for scalability, availability, or both.

4.2.1 The Proxy Front End

All interaction between clients and the proxy occurs through the front end, the only component visible to the "outside world". It is the attachment point to which clients connect for proxy service. Our HTTP proxy implementation is compatible with the W3C protocol standard for HTTP 1.0 proxies. The front end matches incoming requests with the appropriate client preferences profile (currently based on the IP address of the request) and handles preference changes for clients. The front end does not actually do any of the work of servicing the client requests; therefore, it can be

FIGURE 12. Block diagram of scalable proxy internal architecture. Most nodes are connected via both a high-speed SAN (system-area network, such as Myrinet) connection, and a lower-speed "utility network" (such as 10baseT Ethernet) which also includes connections to the outside world. Components include front ends (FE), a pool of workers (W) some of which may be caches ($), a user profile database, a graphical monitor, and a fault-tolerant load manager, whose functionality logically extends into the manager stubs (MS) and worker stubs (WS).



simple and fast, maximizing the number of clients that can be served and minimizing the "turn-around time" before a client request is handled

4.2.2 Workers

The workers are code modules that perform the compute-intensive work of distillation and refinement, caching, and if necessary managing the user interface to the proxy (we discuss UI issues separately below). Our system places minimal requirements of worker structure, making it easy to use off-the-shelf code. A given NOW node may host one or several workers. The workers are "dumb" in that they specialize only in content transformation; the other aspects of distillation—load balancing across multiple instances of a worker, and determining the correct user preferences for a particular distillation operation—occur in other parts of the system, and workers don't need to know about them.

In addition to content transformation, workers can also modify the output datatype of the content. For example, if a client cannot display JPEG images, a JPEG worker could produce output in a different format that the client can understand, perhaps even a proprietary optimized image format. If the client can only display images that respect certain constraints (number of colors, fixed color palette, maximum area, etc.), the worker can attempt to transform the content such that the constraints are met. This tremendous flexibility gives the proxy the ability to serve a wide variety of heterogeneous clients (as we require in Principle 9 from the Introduction), and since each user's preferences are tracked separately, the specific transformations performed will be user-specific (mass customization). We describe separately a specific example application, Top Gun Wingman, that leverages this capability

Besides distillation, a worker is expected to be able to report two quantities, both of which are used by the Proxy Transcoder Manager (described below) for load balancing:

- Its current workload (e.g., what fraction of the time it has been busy since the last report); this information is broadcast using the CB.
- The estimated completion time of a given distillation operation, given the inputs.

4.2.3 Proxy Transcoder Manager (PTM)

The PTM manages the location of workers, distribution of load across multiple workers, spawning of new workers on demand, and providing the assurance of fault tolerance. In addition, the PTM can spawn workers of a given type when there is sufficiently increased load and when a NOW node is available to host it, or when an existing worker dies because of a program fault.

The functionality of the PTM is split across a centralized manager process and stubs in the frontend and workers (the manager stub and worker stub, respectively). The PTM periodically beacons its existence on a coordination bus channel to which the other components subscribe. The use of the coordination bus provides a level of indirection and relieves components of having to explicitly locate each other. When the front end has a task for a worker, the manager stub code in the frontend contacts the PTM, which locates an appropriate worker, spawning one if necessary. The manager stub caches the new worker’s location for future requests.

The worker stub attached to each worker accepts and queues requests on behalf of the worker and periodically reports load information to the manager. The PTM aggregates load information from all workers, computes weighted moving averages, and piggybacks the resulting information on ites beacons to the manager stub (at the frontend). The manager stub caches the information in these beacons and uses lottery scheduling [WW94] to select a worker for each request. If no worker exists to handle the load, the PTM knows how to launch a new one. The PTM can also detect the “death” of a worker and replace it with a new copy.

To allow the system to scale as the load increases, the PTM can automatically spawn new workers on unused nodes. Another mechanism used for adjusting to bursts in load is overflow: if all the nodes in the system are used up, the PTM can resort to starting up temporary workers on a set of overflow nodes. Once the burst subsides, the workers may be reaped.

The PTM represents a centralized approach to load balancing policy: all load balancing decisions are made in the PTM based on information periodically collected from the workers (workers). An alternative design would be fully distributed load balancing, in which workers maintain local load statistics (and possibly exchange statistics with their peers) and the scheduling decision is made solely on the basis of this local information, rather than in a central “clearinghouse”. The standard argument in support of this design is that the loss of an individual worker may lead to degraded scheduling but not loss of functionality, whereas in our scheme the death of the PTM results in the loss of ability to make scheduling decisions at all. We counter that for our architecture, the centralized approach is superior,*provided the PTM can be made fault-tolerant*, for the following reasons:

- Centralized load balancing algorithms are easier to design, easier to understand, easier to debug, and tend to exhibit behavior that is easier to characterize.
- Centralized load balancing allows a flexibility of policy that is much more difficult to achieve with distributed scheduling. For example, during peak usage hours, we can stratify users into “service classes”, giving users in the higher service classes faster service (e.g., by directing their requests to a set of “reserved” workers whose wait queues are kept shorter than average). Implementing such policy changes in a distributed scheduler is difficult and requires changing every component that contains scheduling logic (i.e. every worker). In our system, we can even transparently start a new PTM with a different policy and then kill the old one, without interrupting system operation.

The key to making the PTM fault-tolerant is the use of a*process-peer fault tolerance* design: when a component fails, one of its peers restarts it (on a different node, if required), while cached stale state carries the surviving components through the failure. After the component is restarted, it gradually rebuilds its soft state. In our system, the PTM and front end act as peers for fault tolerance. If the PTM dies, the front end will detect this (through the loss of beacons from the PTM) and restart the PTM. All workers that were originally communicating with the failed PTM will detect the new PTM and register withit, thereby re-establishing communication and rebuilding the PTM's state. Thus the only way to bring the system down is to simultaneously kill the front-end and the PTM. Since these can be made to run on different physical machines, simultaneous death of both is unlikely. Process-peer fault tolerance is one of the techniques used to achieve the **High Availability** goal of Principle 3 from the Introduction.

4.2.4 Caching

The proxy architecture includes Harvest cache workers [BDH+94] partitioned across several nodes. Harvest suffers from three functional/performance deficiencies, two of which we resolved.

First, although a collection of Harvest caches can be treated as “siblings”, by default all siblings are queried on each request, so that the cache service time would increase as the load increases even if more cache nodes were added. Therefore, for both scalability and improved fault tolerance, the manager stub can manage a number of separate cache nodes as a single virtual cache, hashing the key space across the separate caches and automatically re-hashing when cache nodes are added or removed. Second, we modified Harvest to allow data to be injected into it, allowing distillers (via the worker stub) to store post-transformed or intermediate-state data into the large virtual cache. Finally, because the interface to each cache node is HTTP, a separate TCP connection is required for each cache request. We did not repair this deficiency due to the complexity of the Harvest code, at the cost of a slight performance degradation.

Caching is only an optimization. All cached data can be thrown away at the cost of performance—cache nodes are workers whose only job is the management of soft state. In [GB97], we present a detailed analysis of the performance improvements that a typical cache installation can provide.

4.2.5 Video Gateway

The RTP video gateway (*vgw* in Figure 12) can also be controlled via the CB, and we have developed a Netscape plug-in for this purpose. In conjunction with Netscape plug-ins for the MBone video conferencing tool, *vic*, the RTP gateway allows real-time adaptation of video streams. The increasing use of network video and audio was the motivation for Principle 7 (Multimedia Support) from the Introduction, and the RTP video gateway clearly contributes to this goal.

In the original design of the video gateway [AMZ95], much care was taken to separate the user-interface from the computational engine that communicates over the CB. This clear separation between mechanism (engine) and policy (UI) made it very easy to implement a remotely controlled gateway since at the API level, there is no distinction between a local or remote client on the CB. Additional work is now being done to develop the adaptation policy that will enable dynamic adaptation to varying bandwidths over the bottleneck link.

4.2.6 Control Panel

We have implemented a generic Tcl/Tk-based control panel and GUI for monitoring and controlling the entire system. In particular, the various quantities of interest (load, requests per second, etc.) reported by *any* of the system components can be displayed by the control panel, and individual components can be killed. The control panel works by listening and broadcasting on the CB channels our system uses, as described in Section 4.1.

The goal of the control panel is to make it relatively easy for a system manager to administer the scalable proxy, which in turn will make the scalable proxy more likely to be widely deployed once it has made its research debut. The control panel can be used to graphically monitor the state of the system and interactively control it, and will support operations such as notifying the system manager via e-mail or beeper when a catastrophic failure is detected. Since the control panel is scriptable and extensible, we expect it to be reused by other similar projects in the future.

4.2.7 Dynamic Adaptation

An entity not shown in Figure 12 is the *network monitoring* agent. The proxy is already designed to handle each distillation request with separate parameters (each user can maintain a personal preferences profile); this same mechanism can be exploited to provide dynamic adaptation to changing network conditions. The proxy will listen on the Coordination Bus for notifications about network state changes (for example, a drastic change in available bandwidth caused by a vertical handoff). It can then change the distillation profiles for the affected user(s) so that the end-to-end latency is kept roughly constant despite the network change. For example, if a loss of bandwidth is detected, then more aggressive distillation is necessary in order to achieve the same end-to-end latency; if an increase in available bandwidth is detected, we can deliver higher quality content to the user without increasing the latency. This kind of dynamic adaptation illustrates how proxies can play an important role in providing the adaptive behavior we eventually want from all applications, as set forth in Principle 10 in the Introduction.

4.3 Application, Transport, and User Interface Issues

Although the initial implementation of the scalable proxy was designed as an HTTP proxy, the core is general enough to support non-WWW applications. In particular, we have developed a graphical Web browser for the PalmPilot handheld organizer. Our browser, Top Gun Wingman, heavily exploits the proxy architecture; in fact, the browser will not work at all without the proxy, since it relies on the proxy for complete data reformatting into a simple markup format the client can understand. We were forced to build our own client application for the PalmPilot since we could not locate any existing client to leverage, but in doing so we gained some valuable experience, some highlights of which we now discuss.

We have found that such applications are much easier to develop if they can avail themselves of a higher-level application protocol than HTTP.The reason is that in addition to non-trivial performance overhead, HTTP exposes the wrong abstractions to applications and lacks some high-level abstractions that we find useful for adaptive applications of the kind described in this document. For example, HTTP exposes host names and IP addresses (thus requiring TCP/IP semantics), is verbose and space-inefficient, and requires a reliable stream abstraction (TCP) even though its behavior is closer to datagrams than streams (most requests are short and may even fit in a single IP packet, and data from simultaneously outstanding requests may be delivered in an order different from that in which the requests were initiated). Because the protocol is stateless, each HTTP request is accompanied by myriad headers containing authentication information, compression information, and minutiae that must be parsed and consume network bandwidth, even though most applications don’t care about them. On the other hand, HTTP provides no abstractions to support refinement (there is a notion of “quality negotiation” which could be used to crudely support distillation, but it is not well-specified, does not allow fine control over distillation parameters, and is not supported in any current implementation of the protocol), no way to keep session state (cookies only provide session state to individual servers), and no general mechanism for identifying the user’s client device capabilities or tracking and reacting to network state changes.

To address some of these limitations, we have devised a simple application-level protocol by which applications can communicate with the proxy. The GloMop application-frame transport protocol requires a single reliable stream connection using any protocol (such as an error-correcting serial modem). It provides an abstraction of an asynchronous delivery channel of application level frames [***CITE ALF PAPER***], possibly unordered but self-describing. The protocol stack delivers one ready-to-render ADU at a time; thus the client logic is extremely simple--wait for an ADU to arrive, render it, go back to waiting. A simple user interface handles navigation clicks to go to other pages, by sending an ADU encapsulating the new request. Receiving a duplicate ADU is idempotent, and receiving an ADU that was never requested is correct, giving us the ability to support “true push” multicast applications such as the MediaBoard for free. (Although the network stack on the PalmPilot does not support IP multicast, the proxy does support it and can act as a unicast tunnel.)

An obvious drawback of this approach is that client applications must be rewritten (although in the case of Java, a Java client applet could be embedded in a Java-capable browser and exploit existing browser support). In general we have found that if a device already supports the underlying semantics required by HTTP with reasonable performance, as is the case for laptop PC’s, then we should continue to leverage the legacy HTTP clients; but otherwise, for example with current-generation PDA’s, it is far simpler to implement the bare-bones support required by the GloMop bytestream protocol than it is to bootstrap an entire HTTP implementation onto the device. We believe that the widespread availability of a high-level API such as GloMop will ease and encourage application development for a wide variety of clients, supporting Principle 9 (Heterogeneous Clients) from the Introduction.

In addition to bringing the “high availability” requirement into sharp relief, heavy dependence on a proxy that performs distillation and refinement presupposes the existence of a user interface to control these features. For example, in Top Gun Wingman, tapping on an inline image with the stylus brings up a pop-up menu from which the user can choose to refine the image to full screen size, refine it to its original size (which may be larger than the device’s screen and require horizontal and vertical scrolling to view), or follow the hyperlink that the image points to (if the image is a hyperlink). However, comparable functionality is needed for the HTTP proxy for “stock” browsers, where it is not trivial to modify the browser’s UI. For example, browsers already define a behavior for left-clicking on an image, and overriding this behavior in an elegant way is difficult. Recently, client-side extension technologies such as Java and JavaScript have made this task somewhat easier; we have experimented with both mechanisms for improving the UI to our HTTP proxy [FGC+97].

The following table summarizes the most important differences (from the application developer’s standpoint) of exploiting a custom protocol and custom client features as opposed to “stock” clients and HTTP.

Feature	Standard Client	Client with GloMop ASL
Distillation UI	Substitute distilled image for originals, or modify URL’s of images. Both cause cache pollution unless pragmas are used (and observed by client).	No cache pollution since protocol and client have built-in notion of representation vs. document.
Refinement: fetch original	Use Java, JavaScript, or plug-in architecture to provide user interface controls for refinement	Exploit native GUI support, e.g. pen gesture on distilled object opens original (in a new window if desired).
Refinement: zoom in/out		Use native GUI objects to allow user to “drag out” region to be refined.

Page segmenting	Proxy creates new pages on the fly from a single large page, using “magic” URL’s which are specially recognized when passed back to the proxy. Causes cache pollution in client and raises cache management issues at proxy.	Pages are naturally split up into chunks as a foundation of the protocol. It is not necessary to “fool” the client. A table of contents for the pages is automatically generated, also as part of the API.
User preference tracking	User prefs must be mapped from IP address (user re-registers for each new session to re-establish username-to-IP mapping), or else transported in cookies or URL’s.	Initial connection to proxy includes registration handshake, authentication (if desired), notification of which datatypes client can handle, and prefs profile lookup.
Security	SSL; requires changes to server and server registration with a CA; proxy is only allowed to act as a tunnel for SSL	Separate authentication step can use Charon [FG96] lightweight Kerberos-based protocol; only the user and proxy need to share a secret. (Proxy-to-server link would still require SSL.) This will work with all other apps, not just WWW.
Protocol latency reduction	HTTP keep-alive doesn’t work for proxies. TCP setup/teardown is very expensive and adds 1.5 roundtrips of latency per connection. (This overhead is only partially mitigated by the Sessions Layer described in XXXX; it cannot be entirely eliminated.)	Very low overhead (a single stream connection) and high efficiency to the server, including compression that is invisible to the client app.

4.4 Summary

The scalable proxy is the key to providing**Dynamic Adaptation** (Principle 10), which enables support for a broad range of **Heterogeneous Networks** (Principle 1) and for reacting to drastic changes in network performance, such as occur during vertical handoffs. The proxy’s internal architecture is designed to fulfill the following additional goals:

- **Scalability** and **High Availability**: each instantiation of the proxy will support thousands of users by leveraging NOW technology, and will use fault-tolerance techniques such as redundancy and process pairs to mask transient failures in individual system components.
- **Multimedia** and high **Performance**: the ability to do datatype-specific compression (distillation) and refinement enables low-bandwidth users to access multimedia content in ways that were previously impossible, and optimizes for low latency content delivery.
- **Heterogeneous Clients**: since distillation can create output for an arbitrary client device, and we can track each user’s preferences individually, the proxy resources can be amortized across a diverse user population with widely varying clients.

5 Network Services

The Network Services component of the architecture builds atop our innovations in network-layer routing and transport and our use of application-specific proxy-servers. It provides support for locale-specific discovery of available resources (such as proxies) and "on-the-move" reconfiguration of and adaptation to these resources. This functionality is required to support seamless interaction with the environment as clients roam in autonomous (separately-administered) overlayed networks.

Users wish to invoke services -- such as controlling the lights, printing locally, gaining access to application-specific proxies, or reconfiguring the location of DNS servers -- from their mobile devices. But it is difficult to obtain wide-spread agreement on "standard" interfaces and methods for such service invocation. Thus, the challenge is to develop an open service architecture that allows heterogeneous client devices to discover what they can do in a new environment while making minimal assumptions about standard interfaces and control protocols.

The basic premise in developing such a component in our architecture is that providing an "IP dial-tone" to clients isn't enough. We must augment basic IP connectivity with an adaptive*network services* infrastructure that allow users to control and interact with their environment. Distillation proxies are one example of a network service that may require discovery and reconfiguration. We also must provide meta-services that allow for interactions between particular separately-discovered services to allow for interoperability given inevitable variability and lack of "a priori" standards. The challenge is developing an *open* service architecture that allows heterogeneous client devices to discover what they can do in a new environment, and yet which makes minimal assumptions about standard interfaces and control protocols.

In developing this architecture, we have designed, implemented, and deployed in our Computer Science building the following example services:

- untethered interaction with lights, video and slide projectors,a VCR, an audio receiver, an echo canceller, motorized cameras,video monitors, and A/V routing switchers from a wirelesslyconnected laptop computer;
- automatic "on-the-move" reconfiguration for use of localDNS/NTP/SMTP servers, HTTP proxies, and RTP/multicast gateways;
- audited local printer access;
- visualization of physical geography and discovered object locationsvia a protocol for interactive floor maps; \item tracking of users and other mobile objects with "caller ID;"
- advertising available local services (or other data) tounregistered ("anonymous") clients.

In realizing this architecture, we employ a few key techniques:

- augmenting standard mobility beacons with location information,scoping features, and announcements from a service discoveryprotocol;
- using interface specifications that combine an interfacedefinition language with the semantics of a model-based userinterface; and
- hosting scripts in the infrastructure that:
  - map exported object interfaces to client device control interfaces,
  - compose object interactions, and

- automatically remap the destination of object invocations to changing server locations.

The physical components of the testbed in a local seminar room are illustrated in Figure 13. We also leverage facilities in a collaboration laboratory (CoLab) and a student office; all contain devices that can be accessed and/or controlled via our software.

FIGURE 13. Part of the project operating environment: 405 Soda Hall.

The network services' subcomponents are a scoped service discovery protocol and bootstrap, scoped access control and security, and facilities for passing and mapping discovered interfaces to client devices. We start by breaking these down into a discussion of *Controllable Objects*, describing issues in connecting devices to the network and providing them to local users; *Service Advertisement and Discovery*, describing how such objects and services need to be named and advertised; and *Mapping Client Controls to Exported Objects*, where we describe how to make theses interactions seamless. We then dive into implementation issues and details, describe the suite of example ad hoc mobile services incorporated into our testbed, and finish with discussion, related and future work, and conclusions.

5.1 Controllable Objects

Most physical objects provide only manual controls. A *controllable object*, on the other hand, responds to control requests or transmits status information through an exposed interface accessible over the network. Objects must be augmented with this ability to allow for network-based access. Issues in doing so include aggregation of objects into a controllable unit, addressability/naming, and conflict resolution.

5.1.1 Aggregation

At what granularity must controllable objects be implemented? In keeping with our goal of adaption, we allow it to be arbitrary. Requiring fine-grained controllable objects is difficult because individual objects may be too numerous or the expense of individual control may be too high. For example, while it is possible to make every lightbulb its own controllable object, the sheer number of them in a typical building, the expense of assigning processing to each one, the difficulty of wiring each to the network, etc., would mitigate such a decision. Instead, control functionality could be assigned to a bank of lights, and what is augmented is the switch bank rather than all of the individual lightbulbs. In general, the granularity at which object capabilities are exported shouldn't specified by the architecture. The difficulty, then, is to allow client controls to aggregate and subset controllable objects components in a manner transparent to the client interfaces to them.

We provide this feature by providing clients with a facility for hosting scripts that map exported object interfaces to client device control interfaces and compose object interactions. The scripts leverage the use of interface specification languages for object description, allowing access to subcomponents and composition of remote objects. This disassociates controllable object granularity from actual control granularity.

5.1.2 Naming

Another impact of making controllable objects accessible is that the current infrastructure for naming must be extended to include them. These objects do not have individual IP addresses or session descriptions, but instead are accessible through servers and, due to location-based usage, often have fine geographic scope. Users want to make queries based on geographic information (location), data type (position in class hierarchy), scope (accessibility range), and the control authority (the "owner" and/or position in an organization hierarchy for dealing with access-control). These properties can dynamically change, and the hierarchies are not strict (i.e., there can be multiple paths from the root). It is unclear whether these four orthogonal components need to coexist in the globally-visible naming scheme (augmenting or acting together as a fully-qualified unique object name), or whether some can be treated as "properties" rather than elements of the name.

Existing methods for naming include using the Domain Name Service (DNS) [MD88] for objects with unicast IP addresses or the Session Description Protocol (SDP) [HJ97] for lightweight sessions. DNS is a widely-deployed distributed name service. SDP is a container protocol for associating a single name with a collection of application-specific multimedia transports and their (most often multicast) channels. SDP messages are delivered via the Session Announcement Protocol (SAP) [Han97], an announce/listen protocol that uses scoped constant-bandwidth allocations.

Alternatives for the implementation of object naming include extending DNS with new record types, extending SDP/SAP with new application types and finer scoping, hybridizing the two, or developing a separate hierarchy to match this need rather than overloading DNS and/or SDP/SAP. Further implications of this decision are noted in Section 5.2, where we describe service announcement and discovery. Our current proposal for the geographic name axis is presented in Section 5.5.1, where we describe our prototype map service.

5.1.3 Shared Control Conflicts

When multiple users attempt to share a controllable object, there is the potential for conflicts in the requests. Existing systems manage this difficulty by providing well-formed application-specific solutions and limiting the set of conflict states. One example is the elevator. Requests to an elevator are not commands, but are instead idempotent inputs to an algorithm that decides the order for actions to take place (if at all). Individual elevators react to input combinations differently, and this is acceptable. We propose using such application-specific algorithms for controllable objects (encapsulated behind the remote object invocation specification). Leveraging "authentication" features such as locking (minimally coarse-grain, possibly finer) and access levels ("capabilities") can assist in reducing the conflict state set and is very practical; i.e., the “owner” of the device can always override “users,” etc.

5.1.4 Cameras as Object Interfaces

Another approach for interacting with objects is to use video capture augmented with image processing (“computer vision”) where applicable. Example uses of this approach include fine-grain object tracking, directionality sensing, and event triggers keyed to particular circumstances [MF96]. For example, a camera can be used to detect the opening of a door or window. In this case, it is the camera that exports the control interface. Using cameras for such duties has extensive implications for security and privacy control, but is a viable alternative when direct manipulation is not.

5.2 Service Advertisement and Discovery

The function of a service discovery protocol is to allow for the maintenance of dynamic repositories of service information, advertise the availability of this information, and support attribute queries against it.

Service advertisement must scale with both the number of advertised services and to the wide-area. Even as larger numbers and classes of devices become network-accessible (e.g., “IP light bulbs”), the bandwidth consumed by these advertisements must scale sub-linearly. Fortunately, repository information is often local in nature, and though objects can be addressed globally, it can support eventual consistency semantics. This allows for the use of a soft-state update approach such as an announce/listen protocol. These announcements and queries can be scoped and this scoping can provide hierarchy. Difficulties include that the scoping granularity may be very fine, at the level of individual rooms or network subnets/cells, and that scopes must dynamically adapt to support incremental, independent local deployment of various services.

Our observation is that this requires a technique combining the properties of a distributed name service (e.g., DNS) and an announcement protocol (e.g., SAP). The latter “pushes” unknown names/descriptions to the client to facilitate discovery, while the former allows the client to “pull” information about objects given their names. The hybrid technique is to advertise the location of local name services in addition to object descriptions. This allows a single message to, in effect, advertise a collection of objects, and provides advertisement hierarchy (possibly, but not necessarily, aligned to the naming hierarchy like DNS) with scaling sub-linear in the number of advertised objects.

The Service Location Protocol [VGPK97], a resource discovery protocol under development by the IETF Service Location working group, is one proposal for implementing such a service. In SLP, query processing is performed at directory agents and distributed via multicast. Our protocol will leverage features of SLP, specifically the query grammar and message formats.

For basic operation, the only mechanism necessary is a function to allow mobiles to map names to values. These can be obtained by querying a local server or by receiving them via multicast (our current prototype operates via the former.) We describe our own mechanisms for finding the correct local server or multicast addresses and initializing the mappings. Finding one of the correct local servers is similar to delivering the correct*scope* attribute to the mobile host in SLP. (The SLP *scope* attribute is used to administratively aggregate an otherwise disparate set of services.)

5.3 Mapping Client Controls to Exported Objects

The network services component allows for UI widgets to remain in familiar locations and in a familiar form even as the particular devices that the widget controls change (e.g., a light switch). We now describe our mechanisms for enabling this behavior.

5.3.1 Transduction Protocols

A transduction protocol maps a discovered object interface to one that is expected by a given client device. It supports interoperability by adapting the client device's interface to match the controllable object's interface. It allows for custom user interfaces to ad hoc services, such as allowing a virtual "light switch" on a control panel to always control the closest set of lights. Without a mapping function, every change in location might require a new interface be retrieved.

An issue with transduction protocols is how to map control functions into a UI supported by the portable device. As an example, assume a client device has a two-position switch widget for use with the local light controller. At a visited location, the light controller supports continuous dimming. In this case, the client may substitute a slider widget for the switch. If it cannot do this (or chooses not to), then the purpose of the transduction protocol is to map the on/off settings of the UI to one of the two extremes of the actual dimmer control.

To support interoperability, we allow services to transfer an entire GUI to the client in a language it understands, avoiding the need for transduction. (This similar to the Java applet usage model but with multiple language support where necessary.) Whenever possible, though, we augment the GUI (or replace the GUI completely) with an interface specification (described in Section 5.4.8). Through the interface specification, the system discovers the two data types that need transduction. This allow the mapping function to be inferred (heuristically, from a library, or by querying the client) and then installed at the local transducing proxy that sits between the two endpoints. The interface specification can also be used directly to generate a rough GUI when no interface implementation appropriate for the client is available, or when only portions of the controllable objects' interface is of interest of the user (i.e., to conserve screen real estate or to add a button into a user-defined control panel).

The interface descriptions not only allow for data type transducers between client and server; they also provide the critical layer of indirection underneath the user interface. Examples include composing "complex" behaviors and remapping the destination of object invocations to account for mobility.

5.3.2 Complex Behaviors

Objects have individualized behaviors. We wish to couple and compose these individual behaviors to obtain more complex behaviors within the environment. For example, consider a scenario where music follows you as you move around a building. One behavior of the sound system is to route music to specific speakers. A behavior of location tracking services is to identify where specific objects are located. A "complex" behavior allows us to compose these more primitive behaviors of sound routing and location tracking to obtain the desired effect of "music that follows."

A key problem is that there is no common control interface for individual components. Furthermore, some behaviors may require maintenance of state that is independent of both subcomponents. An example of the latter is instructing the coffee maker to brew only the first time each morning that the office door opens. Another issue is a policy-level difficulty implied by this scenario: resolution of incompatible behaviors. If another user considers music to be noise, the visiting user's music may or may not be turned off in their presence, depending on seniority, social convention, explicit heuristics, or otherwise. At a minimum, the system must guarantee that it will detect such incompatibilities and notify the user(s) involved in order to avoid instability (e.g., music pulsing on and off as each individual behavior is interpreted).

Once again, as in transduction, our solution is to use *interface discovery* (learning new objects' input/output data types), paired with the data type transducers (for manipulating those data types) to allow objects to be cascaded to achieve the desired complex behaviors. Additionally, we supply intermediate entities (proxies) that maintain state that is independent of the constituent subcomponents. This allows for the incorporation of such features as conditional statements and timing information.

5.4 Implementing Service Interaction

This section describes some of the implementation details of the services architecture subcomponent.

5.4.1 Basic Operation

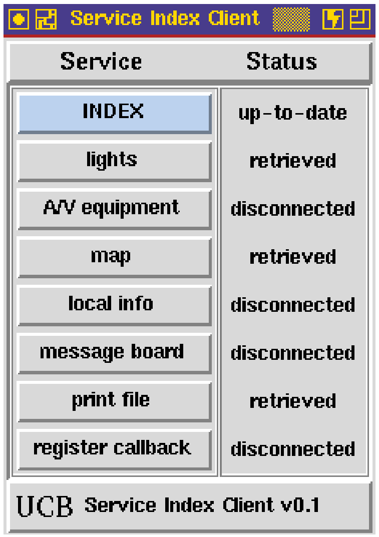
The prototype allows a mobile host to enter a cell, bootstrap the local resource discovery server location, and acquire and display a list of available services. It also allows users to maintain a database of client-side scripts to be executed when particular services are discovered for use in reconfiguration, local state updates, and to trigger location-dependent actions. Similarly, a set of scripts are maintained in the infrastructure at each site for locale-specific adaption such as transduction and composition. The prototype also allows for simple, incremental addition, deletion, and modification of available local services.

The key components of the compete system are the "service interaction proxy" (SIP), the "service interaction client" (SIC), and the "beaconing daemon" (beacond) programs. These prototypes implement and integrate selected infrastructure components of our overall mobile services architecture. The SIC runs on the client device and provides the base functionality for discovering and managing services. SIPs run at domain-specific granularities and aggregate a group of services with a single set of advertisements. The SIPs also manages the proxies used between the client devices and the individual services. Beaconing daemons run at each base station and are affiliated (not uniquely) with the SIP it is advertising.

An example SIC screenshot is shown in Figure 14. SIP and beacond use configuration files and command-line switches, and thus user interfaces are not shown.

5.4.2 System Setup

Each SIP process maintains a database of the services and service elements that it provides to mobile hosts. An example startup file for such a database is listed in Figure 15. It contains three types of entries: *services*, *values*, and *properties*. *Values* are used for generic (key, value) lookups. These are useful for, e.g., detecting the need to update server addresses. *Services* and *properties* are used to specify what, where, and how services are available from that particular location. Each *service* has a unique name, and maintains *properties* such as the version number, a pointer to an associated ISL file (described in Section 5.4.8), pointers to particular language implementations of user interfaces for the service, and the geographic location (if any) for use with maps. *Values* and *properties* may just be pointers to another SIP, allowing simple incremental deployment to subdomains and yielding a notion of topology.



Service	Status
INDEX	up-to-date
lights	retrieved
A/V equipment	disconnected
map	retrieved
local info	disconnected
message board	disconnected
print file	retrieved
register callback	disconnected

UCB Service Index Client v0.1

FIGURE 14. A screenshot of the Service Interaction Client. The SIC is currently a series of buttons that can be used to retrieve and invoke application interfaces.

```
set NAME {
        Soda 405: High-Tech Seminar Room
}
set SERVICES {
        INDEX lights {A/V equipment} map printer {location tracking}
}
set VALUES {
        DNS {128.32.33.24 128.32.33.25}
        NTP {orodruin.cs.berkeley.edu barad-dur.cs.berkeley.edu}
        SMTP {mailspool.cs.berkeley.edu}
        ...
}
set PROPERTIES {
        lights {ISLfile ../helpers/lights.isl version 0.01 \
                location {132 210} appName-tk ../helpers/lights.tk \
                appArchive-tk ../helpers/405/lights405.tar.uue
                appName-tcl ../helpers/lights.tcl \
                appArchive-tcl ../helpers/405/lights405tcl.tar.uue}
        {A/V equipment} {ISLfile ../helpers/htsr.isl location {132 180} \
                version 0.01 appName-tk htsr.tcl \
                appArchive-tk "../helpers/405/HTSR.tar.uue"}
        ...
}
```

FIGURE 15. An abridged SIP services database example

5.4.3 Message-level Detail

The client enters a cell with a beaconing daemon. The daemon sends periodic broadcasts that contain the bootstrap address and port number of that cell’s SIP. The client registers with the base station to establish IP connectivity if it needs to. It then requests the well-known meta-service *index,* which returns a list of the services available. Based on the contents of the reply, the client renders labelled UI buttons for unknown services, executes scripts in a database to allow for locale-specific reconfiguration, and tells the local SIP to remap the location of running services and setup any necessary widget binding remappings.

When a user wishes to use a particular service, the client software checks its local cache of applications. If an interface supporting the requested application is not there, it asks the SIP for the service’s “properties.” This is a list of available interface descriptions and/or implementations. It also receives any service metadata (such as version numbers). It then chooses either to download a particular interface implementation (e.g., as a Java applet), the generic interface description, or both. The SIC then unpacks the received archives, sets up transducers matching the interface description to the device characteristics, and finally executes the GUI.

An example exchange of protocol messages for a client moving between SIPs is illustrated in Figure 16.

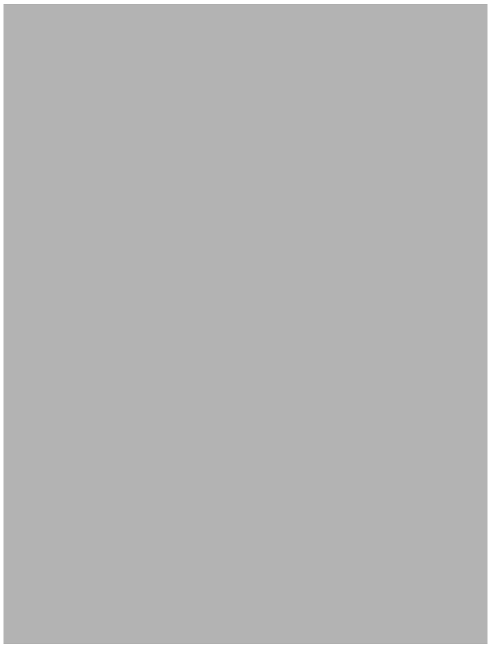


FIGURE 16. Protocol message timings for a client moving between SIP servers (dashed lines are beacons): (a) index #1 request/reply (b) request/reply for “lights” ISL file and interface (c) index #2 request/reply (d) “lights dim” button press retrieves new ISL file to remap RPC, then completes.

5.4.4 Client Bootstrap

For a client to use services, it must first find the address of the local resource discovery server or the local multicast address where services are advertised. In our architecture, this bootstrap above IP is minimal: there is an indirection embedded in the mobility beacons. This minimal bootstrap standardizes the mechanism without constraining its interpretation, thereby allowing variation in resource discovery protocols as they evolve.

5.4.5 Beaconing

Beaconing is required in a system to facilitate notification of mobility-based changes in the relative position of system components. Its use is motivated by inherent availability of physical-level hardware broadcast in many cellular wireless networks and the need to track mobiles to provide connectivity.

Two issues arise once the decision to beacon has been made. The first is which direction to send them: uplink, downlink, or both. The second is what information to put on the beacons, if any at all. (An empty beacon acts as a simple notification of the base station address, available in the packet header.) These issues are discussed in the following subsections.

5.4.5.1 Beaconing Direction

In terms of choosing whether to have client devices or infrastructure servers beacon, existing systems can be found which have made either choice. Client beaconing is used in both the Active Badge [HH94] and ParcTab systems [SAG+93], while server beaconing was used in Columbia Mobile IP [IDM91]. IETF Mobile IP utilizes both periodic advertisements*and* periodic solicitations.

One might expect that the different policies optimize for different applications' operating modes. This is indeed the case: there are trade-offs in such a decision, as it varies allowances for privacy, anonymity, particular protocols' performance, and scalability.

There are a number of metrics for considering qualitative trade-offs between the two decisions:

- *Power:* Less power is consumed at the mobile by periodically listening than by periodically transmitting, but this difference can be mitigated by hardware/MAC design [SGHK96].
- *Detection:* When base stations (BSs) beacon, mobiles need not transmit to detect when all contact is lost. When clients beacon, BSs need not transmit to detect user mobility.
- *Multiples:* With BS beaconing, detection of multiple beacons can be used to assist handoff. With client beaconing, the number of received beacons specifies the number of clients in the cell.
- *Location Anonymity:* When BSs beacon, anonymity is preserved for non-transmitting mobiles; when clients beacon, the granularity of the infrastructure is invisible to users.
- *Geographic Mapping:* BS beaconing maintains a consistent mapping between geography and beacon broadcast cell; client beaconing maintains a mapping of clients to multiple cells.
- *Bandwidth Scaling:* BS beaconing implies less beacon traffic per cell given a natural many-to-one mapping of mobile hosts to base station cells. Conversely, client beaconing optimizes for very small cells. (Assuming other parameters remain constant.)

Our system uses base station beaconing. We believe this is the correct design choice for three key reasons: the support for user (rather than infrastructure) anonymity, better bandwidth scalability in a network where there are many MHs per BS, and because power is more precious on mobile devices.

5.4.5.2 Beacon Augmentation

The second question is whether to augment mobility beacons with additional data. Augmenting beacons with application-specific data does two things. It makes data available to mobiles *before* registration (in the Mobile IP sense), allowing the possibility of “anonymous” access to this broadcast data (at a cost of management overhead and increased beacon size due to the piggybacking). It also aligns the mobility beacons with a form of announce/listen protocol that has limited announcement timer adaptability. The announcement timer can only be set to discrete multiples of the base beaconing rate (that rate determined as sufficient to detect handoff within some acceptable latency.)

Possible uses for such piggybacked beacon data include:

- merging of other periodic broadcasts to amortize headerand MAC overhead (e.g., NTP beacons, Mobile IP foreign agent advertisements);
- pricing information useful to the host to determine whether toregister;
- commonly accessed time-variant data; \item a list of some or all of the available services in the cell;
- “tickets” for providing scoped access control (discussed in Section 5.4.7).

The utility of beacon payload augmentation is highly dependent on the direction of the beaconing, traffic patterns, and application mix. An argument against augmenting beacons at all is that orthogonal applications shouldn’t mix their data units that may have been “properly” sized by the application (c.f., application-level framing [CT90] or joint source-channel coding [MVV96]).

We choose to augment our beacons with bootstrap information, a ticket for scoping of services, and a dynamically configurable application-specific payload. The encoding is shown in Figure 17. One common application-specific payload is the contents of the cells’ *index*, allowing anonymous assessment of available services and reducing discovery latency.

FIGURE 17. The service beacon encoding includes bits for the service interaction bootstrap and location queries. Not shown are the details of any particular mix of application-specific payloads.

Whether merging data into beacons is a benefit depends on the metric of evaluation. We are still trying to quantitatively determine which data, if any, is best dedicated to these bits for optimizing reasonable client-driven workloads.

5.4.6 Charon: Proxied Secure Access

Authentication to network services is based upon the Kerberos IV [Ste88] infrastructure. We have engineered a partitioning of the Kerberos implementation, called Charon [FG96], for use between resource-poor clients (e.g., PDAs) and a proxy. This partitioning allows the proxy to interact with the Kerberos infrastructure on behalf of the clients, giving the clients full authenticated (and optionally encrypted) access to the services but relieving them of the burden of having a full Kerberos implementation. Because the Kerberos infrastructure itself is unmodified, all of the security features of Kerberos are preserved, and resource-capable clients can make use of Kerberos as they normally would.

Charon is our proxied implementation of Kerberos that provides indirect authentication and secure communications with personal mobile devices. By indirect authentication, we mean that most of the computational resources needed to conduct the Kerberos protocol and establish a secure channel with a network service are located at a proxy, a process running on a resource-rich desktop workstation in the well-connected infrastructure. This proxied approach simplifies the client software considerably: minimal client-side software is sufficient to preserve Kerberos semantics and security guarantees, while off-loading substantial protocol processing, credentials management, and other housekeeping tasks to a proxy in which minimal trust must be placed.

Charon provides three secure-access models:

- A means of authenticating clients to a service attachment point (proxy) using a Kerberos-based protocol;
- Establishment of a secure communication channel to that attachment point, on which future logical channels can be multiplexed, thus providing encryption to the proxy when link-level encryption is weak or unavailable;
- Both end-to-end and proxied access to two-way-authenticated Kerberized services in an existing Kerberos infrastructure.

Only DES encryption and decryption need be ported to clients; in our environment these clients may suffer from limited resources and dissimilar development environments, both of which would make a full port of the Kerberos implementation exceedingly difficult. Furthermore, rather than understanding multiple message formats and communicating with multiple Kerberos entities, the client needs to understand only a single message format and communicates exclusively with a single entity (the Charon proxy), which can also serve as a network gateway if necessary.

Neither the user’s Kerberos password nor the key required to construct Kerberos authenticators (used when requesting access to new services) ever leave the client. Charon has the same immunity to protocol-based attacks as Kerberos does, and is more immune to certain end-to-end attacks because of the nature of the devices on which it is designed to run, providing an alternate means of Kerberos authentication for security-conscious users.

5.4.7 Scoped Access Control

Making services available to visitors brings up a host of general security issues, including those specific to the wireless domain [Bro95,AD94]. In addition to standard cryptography-based security with passwords, capabilities (e.g., Kerberos), and public-key encryption, service interaction systems specifically require additional access control. This is due to our extending devices to be network-addressable entities. In general, global access control is necessary, but not sufficient; the expected behavior that environmental changes can only be affected by people in that environment (e.g., lights cannot be turned off by a person across the country) has been broken. Maintaining this norm is important when extending existing human social metaphors into an environment with controllable objects. We address this by embedding *tickets,* random fixed-length bit vectors, in the mobility beacons and requiring the current ticket to be included in all communications to servers. Periodically changing the tickets in an unpredictable way (truly randomly) and scoping the broadcast (implicitly via the cellular wireless network broadcast cell or explicitly with the IP multicast TTL field) prevents remote access from nodes *even on the access control list* that aren’t local. This pairs the geographic scoping of the environmental controls (what we cannot control) to the topological scope (what we can control). This ticket-based exclusion can be overridden (by separately multicasting or unicasting the ticket when necessary), but by making the default access restricted, we better emulate the existing paradigm.

5.4.8 Client Interfaces

5.4.8.1 Motivating Interface Specifications

Clients can be computationally impoverished, have variations in display (color depth, resolution, screen size), support different interface paradigms (keyboard, pen, touch), and are often interchangeable with one another (and therefore not preconfigured).

Due to the need to support such end devices, especially extremely resource-poor PDAs, our architecture focuses on providing thin client interfaces and offloading processing to proxies. Recognizing that expecting custom UIs to be available for all services on all different types of hardware isn't realistic, we propose exposing controllable objects through an *interface specification language* (ISL). The ISL is used *in addition* to other reference language implementations of an interface, thus allowing for compatible but independent coexisting interfaces. It exposes the syntax of each services' control interface. Upon discovery of a service, the client device checks to see if a language implementation is available that it can support, and if not, uses the ISL file to learn the RPC calls and parameters that can be used to access the service. It additionally allows the device to adapt the representation to a format appropriate for the device's characteristics, and allows the user to place the elements manually and independent of one another for fine-grain control. Automatic layout heuristics can also be used.

5.4.8.2 Interface Specification

The conventional notion of an Interface Definition Language (IDL) (e.g., the CORBA IDL) is to specify parameters, parameter data types, and parameter-passing conventions for remote object invocation [Cor97]. The basic function of a Model-based User Interface [SFG93] is to specify interfaces as structured widget hierarchies along with sets of constraints. The actual interface is derived from the model at run time or through a compilation step. This allows interfaces to maintain a consistent look-and-feel even as elements are added and deleted. Our goal is to unify these approaches, allowing remote object APIs to be augmented with UI models. This hybrid would allow both composition of the object invocations (RPCs), and separate subsetting/aggregation of object UI elements. We call this combination an interface specification, and its grammar an *Interface Specification Language,* or ISL.

As a concrete example, we would like to allow an application to buffer DNS queries (calls to one object method) through a local cache (calls to another object), while instantiating the new functionality either through the original UI or via a new one.

An important design criteria is to make the hybrid robust enough to be applicable to a variety of classes of client devices (each with their own implicit assumptions about widget implementation), yet simple enough to be manageable.

The use of an ISL requires services to explicitly support a layer of indirection through the service discovery mechanism, allowing transparent remapping of individual interface elements. This indirection is critical for allowing services to be composed. As an example, the RPC command to spawn an audio conference can be rerouted to a client script that first reduces the volume of the room's music player and *then* passes along the original RPC.

Our current implementation has interfaces manually implemented in Tcl/Tk and Visual Basic. Currently, ISL uses an ad hoc grammar tuned to Tk. A related developing protocol with very similar needs is the Universal Remote Console Communication Protocol (http://trace.wisc.edu/world/urcc), motivated by research in supporting interfaces for disabled users.

5.4.8.3 Prefetching

As an optimization, clients can prefetch the ISL files for active services. We illustrate with a concrete example from our prototype. As the user moves between rooms, the light controller application UI remains the same. When the user changes the lighting in a new cell, the client application sends the new SIP a request for the lights ISL file, enabling the RPC command invoked by the existing interface to be remapping so that the recipient will be the new server. This late-binding is used to conserve bandwidth on the wireless link; the total number of ISL files may be large and the client may use one only infrequently.

The problem with late-binding is that this entire operation latency seen by the end user; in practice it can be perceived as a possible error condition. (The button "doesn't work" for a number of seconds after it is invoked, and for this period it should probably be greyed out in the UI.)

This delay can be minimized by transparently remapping the interface elements to the new server as soon as possible. To do so, we add one bit of per-service state, "active vs. inactive." This flag is set to "active" whenever there is an RPC call from that service, and reset to "inactive" by a timeout. Upon receipt of any beacons with a new SIP, services with the "active" bit set (and available in the new location) have their new ISL files prefetched automatically. (In our current implementation, the *index* meta-service is always prefetched.) Delays can be further minimized through mobility prediction [LM96], allowing prefetching in response to assumptions about user mobility patterns.

Prefetching is important in this domain because the delay experienced by an end user using a high-latency, low bandwidth wireless interface can be substantially less with prefetching and transparent remapping than with demand-paging.

5.4.8.4 Client-side Security

As for mobile code security, by transferring only an interface to the client, it is probable that a sandboxed environment (such as Java, Safe-Tcl, or Janus [GWTB96]) can be used without constraining the service's functionality. This is another benefit of the proxy-based access model: it segments the security domain and thereby allows potential internal holes to be screened from the user. This is quite similar in principle to Java's restricting applet communication to the applet provider.

5.4.9 Naming Scheme

We express controllable object names as a globally-unique fully-qualified object name ("FQON") and a tuple of properties. Some properties are required while others are optional but standard. (Service-specific properties are also allowed, of course.) The set of required properties includes:

- geographic location as a name hierarchy of maps that contain the object
- the data type of the object as a class in a class hierarchy and a version number
- the name of the "owner" of the object to indicate where and how to obtain authenticated credentials (if necessary)
- a pointer to the controlling server process (machine/port)

A common but optional property is a set of pointers to "peer" controllable objects and tags that note the I/O data flow that makes them peers. These are useful for specifying, for example, that the (analog) input source to a television monitor is a particular A/V switcher.

We not as of yet incorporated scope information into the object properties, but like the rest of our proposed naming scheme, this remains as critical area for future research.

5.5 Prototype Mobile Services

In addition to the prototype service discovery and interaction implementation, we have developed a number of services using the framework. These include maps that specify discovered objects' positions, autoconfiguration, location tracking with privacy allowances, audited printer access, and interfaces to audio/visual equipment. We now describe each in turn.

5.5.1 Maps

Given a widely distributed service interaction system supporting very fine-grained services, management of even the subset of information available to the client becomes non-trivial.

We have experimented with using maps for explicit management of these services at multiple locations. Map content is separated into three domains: network connectivity (topology and link characteristics), physical geography (object locations and floorplans), and administrative domain (access rights, pricing, hierarchy).

Our prototype, of which an example view is shown in Figure 18, focuses on physical geography: allowing objects to note their location on multiple overlapping maps and receive requests passed through the map interface (via a button press on the service indicator on the map).

The map protocol itself is a prototype based on using absolute positioning. It is designed to allow objects to position themselves without knowing exactly which map(s) the user is using. It is designed to allow maps to maintain hierarchical ("containment") relationships in a distributed, extensible manner through the absolute positioning information. Represented objects must be sized in accordance with the scale of the map(s) they are located on.

Currently, the map protocol hierarchy is defined to mirror the service interaction proxy hierarchy: each SIP contains a single pointer to a local map. The map hierarchy thereby maintains the same characteristics as the SIP hierarchy: it is can be arbitrarily nested and extended to subdomains without affecting other maps or requiring objects to relocate themselves.

Each SIP database contains pointers to local maps (and potentially different encodings of the same map) and to map metadata including the positioning information (latitude/longitude coordinates of two corners) and peers. This technique extends the technique used for notation of physical geography in proposed "LOC" DNS record type [DVGD96] to objects without IP addresses.

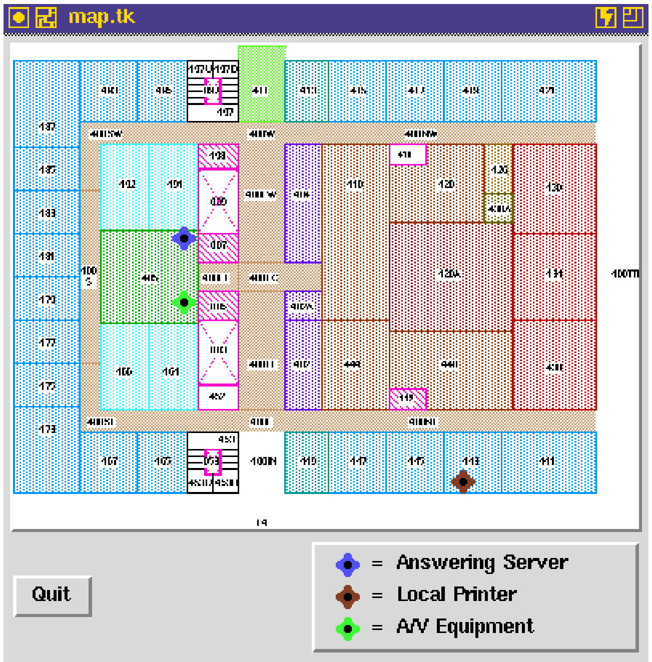


FIGURE 18. Map with discovered object locations, configured for a user in the RF cell including room 405. Clicking on entries spawns the interface to that entity.

5.5.2 Proxy and Gateway Autoconfiguration

Proxy and gateway autoconfiguration is a lightweight service built atop the server reconfiguration service. The difference between it and server reconfiguration is simply that proxies and gateways run in the infrastructure, between client and server. Thus, the infrastructure needs to explicitly support spawning of these entities. Examples of such useful intermediate agents include web proxies that perform on-demand dynamic transcoding [FBGA96], network data caches, real-time media transcoders [AMZ95], and multicast-to-unicast gateways for multicast-unaware client devices (i.e., most PDAs).

Proxy, gateway, and server autoconfiguration is important in a mobile environment for more than just efficiency. Using the “best current practice” technique of hard-coding DNS servers either as */etc/resolv.conf* entries or in the Windows registry, if a user were to move from a location behind a firewall to one that is not, all lookups will fail until an out-of-band technique is used to find a new server and the entry is manually updated. The Network Time Protocol is dependent on server location due to its use of RTT estimation, and is therefore especially suitable for use with automatic reconfiguration. A failure to keep accurate time can break some security systems, notably Kerberos. Spawning a local RTP gateway/transcoder for unlayered data in MBone sessions may be necessary if movement has changed the bottleneck link to a source or to facilitate local management of inter-session bandwidth sharing [AMK97].

Autoconfiguration also adds a level of fault tolerance. If a network link goes down, SIP beacons coming across the failed link will stop. The client will wait for other beacons to be obtained (c.f., overlay networking vertical handoff), and reconfiguration to the new servers will happen transparently.

Our current implementation simply allows for callbacks to be set that track *value* entries in the SIP databases.

5.5.3 Location Tracking

Location tracking is addressed in other systems [ST94,HH94], but these systems suffer from the limitation that client devices must be turned off or not carried to ensure privacy. Instead, users should be comfortable that even while they maintain *continuous* access ("any time, any where"), they can be assured they are not vulnerable continuous detection (e.g., while in a restroom).

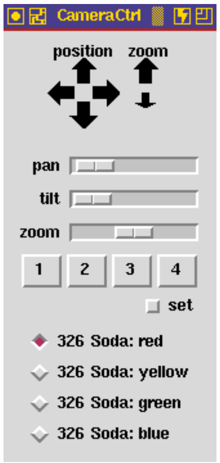
To address this difficulty by using a beacon payload extention. A "caller ID" feature is implemented by piggybacking location queries (consisting of the name of the requestor, the name of the requestee, and an address and port for the reply) on the beacons, allowing the client to reply only if it desires. This feature allows selective exclusion of individual location requests. This would useful, for example, with small children: their device can be configured to only respond to location queries from their parents.

5.5.4 Printer Access

One of the most common examples in the resource discovery literature is local printer access. In our implementation, after the discovery protocol finds the local printer and notes it on the map, clicking on it (or on the "print" SIC button) pops up a dialog box that can be used to send a client's postscript file to the local print server. The server then checks the data, logs the request, prints the file, and returns any status and/or error messages.

5.5.5 Motorized Cameras

We have built software to control both the Sony EVID30 and Canon VCC1 motorized cameras.. The unified user interface to the cameras is shown in Figure 19. Operations supported include pan, tilt, zoom, speed controls, software presets, and combining multiple camera controls into a single applet.



● 326 Soda: red
○ 326 Soda: yellow
○ 326 Soda: green
○ 326 Soda: blue

FIGURE 19. Screenshot of the user interface to the camera controls.

The camera controls are used extensively by members of a current UCB class ("CSCW using CSCW") with remote participants . They use the controls to allow better monitoring of whoever is speaking and to frame a group of speakers into a single view

5.5.6 405 Soda Room Interaction

The "high-tech seminar room," where weekly MBone broadcasts of the Berkeley Multimedia and Graphics Seminar take place, is equipped with a variety of equipment: two slide projectors, a light controller, a video projector, a VCR, a receiver, a DEC workstation, and an Intel PC. All the devices are attached to an AMX corporation control switcher or routed via an AMX router. The DEC workstation talks to the AMX via a RS-232 serial connection, which allows the workstation to act as the control interface.

5.5.6.1 Design and Architecture

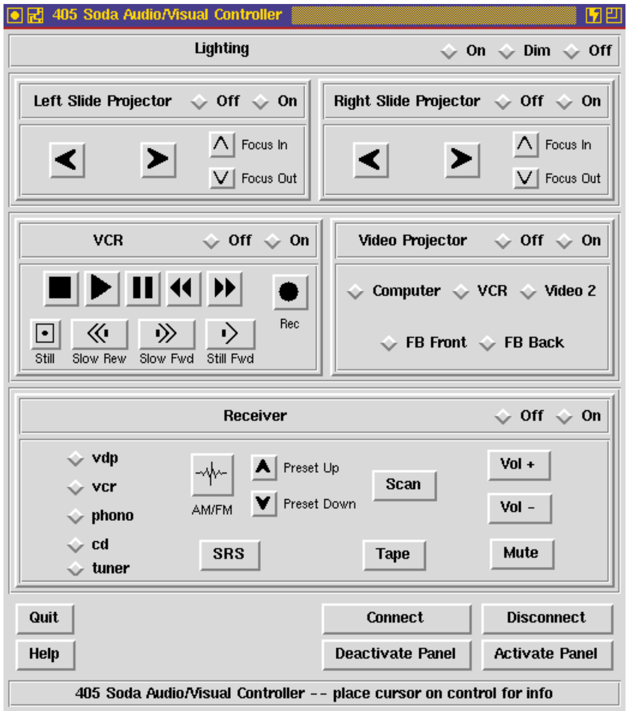
The application, like others in our architecture, is built using the principle of *application partitioning* [Wat94]. Due to the potential lightweight nature of clients, the server is required to bear the brunt of the effort to support fault tolerance, access control, and other such duties. Features can be added to make the internal system interactions more robust with little or no change to the client-side code.

The server runs on an extended Tcl/Tk wish shell which includes the base AMX functions. The server opens an RPC socket and listens for requests to convert to AMX commands. It is also responsible for maintaining the hard state of the system. This leaves the clients free to act as only a UI and cache for soft state.

5.5.6.2 Room Interfaces

Our initial implementation of the room controls includes two separate monolithic Tcl/Tk programs for the room's control, one a superset of the other. The first handles only the lights, while the second handles the most useful buttons (showing them all is excessively complex). The latter interface is shown in Figure 20.

It was these implementations that led us to observe the utility of functional inclusion and the need for variability in the interfaces. It also led us to realize that independent objects should be composable. With such a design, users could create unique UIs that makes the most sense for themselves by leveraging the scripting language and ISL. We are working on allowing the user to maintain sets of service elements by manipulating the interface specifications transparently, for example by dragging-and-dropping individual elements to and from a toolbar.



405 Soda Audio/Visual Controller -- place cursor on control for info

FIGURE 20. Screenshot of the monolithic user interface to the A/V equipment.

5.5.6.3 State Management

Ideally, requests to the AMX could be idempotent, and no state would have to be maintained in the system. However, by the nature of the equipment to which it is attached, AMX requests are *not* idempotent, and cannot be coerced into idempotent versions. For example, if we want to turn on the receiver, the only request we can give is equivalent to "toggle receiver power," which may very well turn the receiver off. The only way to know the effect of this request beforehand is if long-lived state variables are maintained.

Because bandwidth between the client and server is often valuable, state variable updates need to be minimized. Our server is responsible for maintaining consistent state between and during client sessions. Clients are responsible for querying the server for the state info upon connection initiation. For consistency, clients are required to send an update request to the server to ask for state changes. Clients may only commit the change upon receipt of an acknowledgment.

Another issue is dealing with inconsistencies due to manual events. Devices are unable to inform the AMX when a user presses a button on their front-panel. If a user wants to insert a video cassette into the VCR, he or she must first turn it on; the AMX does not register this manual event. Since this is such a common problem, we accept that, at times, the state will become corrupt. We equip the user to correct such inconsistencies via the two buttons at the bottom of the client interface labelled "Deactivate Panel" and "Activate Panel." Whenever a discrepancy in the state occurs, the user can deactivate the panel. All the state-related buttons will then only modify the state variables (on both the client *and* the server) and *not* make requests to the AMX. This means the user can reconcile the information on the panel with reality, then reactivate the panel and once again right the system to a consistent state.

5.5.7 326 Soda Room Interaction

The “MASH Collaboration Laboratory,” or CoLab, is a fully outfitted room designed for extensive use with collaborative multicast applications. The room’s equipment includes a Xerox LiveBorard, four television monitors, four motorized cameras, an A/V routing switcher, an echo canceller, four PCs with output scan-converted and fed into the switcher, a VCR, an audio receiver, an infrared repeater for mimicking remote control button presses, and controllable power switches connected to a lamp and “ON AIR” sign. Weekly broadcasts of a Fall 1997 UCB course entitled “Computer-supported Cooperative Work Using Computer-supported Cooperative Work” are sourced from here, and remote participant feeds are sent to the rooms’ four monitors and audio receiver.

Though the equipment is different from that AMX-based system in 405 Soda, many of the capabilities and issues raised are similar and we will not repeat the discussion here. Only a subset of the room’s media stream control functionality has been integrated into our architecture (camera controls and A/V switching).

5.6 Discussion

A layered view of our architecture is presented in Figure 21. This representation exposes how the various mechanisms described in this paper interrelate. It also illustrates how alternative mechanisms could replace the particular ones we have chosen without affecting the overall service architecture. For example, if some to-be-determined Service Location Protocol *scope* delivery mechanism were to replace our augmented beaconing mechanisms for location management, the interface discovery and data type transduction could remain unaffected.

The lowest layer is the *Announcement* layer. It lies directly above the network and transport layers and implements the basic service bootstrap mechanisms. This includes the embedding of local server information in the beacon payload, the ability to implement scoping mechanisms through the indirection gained by the beacons, and the possibility for additional application-specific payload augmentation as a performance enhancement. Its most basic function is to *find* servers and users.

The *Query* layer uses server location information from the announcement layer. It adds the ability to interact with found entities such as resource servers and service interaction proxies, and provides a structure for attribute queries (e.g., requests for a device-specific service applet interface). Its most basic function is to allow entities to *talk to* other entities.

The *Interface Description* layer is built on whatever query protocol is exposed by the query protocol layer. It defines the set of possible interface descriptions and their semantics. Its most basic functionality is to *map between* the client device interface and the interface advertised by discovered objects.

The highest layer is the *Application* layer. It uses the interface description language exposed by the interface description layer. It encapsulates application-specific state or data not captured by the lower layers. Examples include attaching semantic meaning to particular names (i.e., “camera”) and defining relationships between data values (i.e., the map protocol hierarchy).

FIGURE 21. A layered view of the mobile services architecture.

5.7 Related Work

The Rover [JDT+95] and Wit [Wat94] systems also recognized the need to split applications into a lightweight front-end and more heavyweight proxy at the last hop wireless link. Rover allows the pieces that comprise the partitioned whole to migrate between these two points, but the implemented prototype applications generally only exploit this for moving application data units such as mail messages, news articles, web pages, or calendar entries.

The Service Location Protocol (SLP) [VGPK97] is an example resource discovery and service registration mechanism that can also function as a fine-grained name service. We are interested in moving our resource discovery mechanism over to this evolving Internet standard. Open issues include its undefined “local” scope designation and lack of an explicit scope hierarchy and peering equivalent to our use of pointers in a service database. Our mechanism for dynamically updating the current SIP location could be adapted as a scope discovery mechanism and coexist with other such mechanisms in the proposal (i.e. having a*scope* DHCP option).

The seminal ParcTab [SAG+93] and Active Badge [HH94] systems, along with related work by Schilit [SAW94,ST94,Sch95], were among the first to attack the issues of client applications and network support for mobility in tandem. We borrow much from this work, including the focus on mapping, event notification, and support for impoverished devices. There are some key differences. We support distributed servers, rather than a centralized repository. We employ discovery mechanisms, interface code mobility, and generalize to heterogeneous devices; these are unnecessary in their local-area, homogeneous environment with pre-installed custom applications. We use server beaconing rather than client beaconing, and allow the beacons to bootstrap resource location, define scope, assist fault detection, and provide for some location management.

A transportable X display [WRB+97] is a variation on interface code mobility; it moves users' *existing* interfaces as they move, not unknown applications' interfaces or interface descriptions. It has the advantage that applications need not change at all, but suffers from the limitations that 1) it doesn't support transformations of the interface to formats more suitable to particular client devices, and 2) it does not expose a layer of indirection underneath widget invocations.

The Mobisaic [VB94] and Dynamic Documents [KPT94] projects support a HTML-based structure for varying, location-dependent interfaces. Our scheme generalizes these approaches by incorporating resource discovery and aggregating/subsetting different interface elements.

The Georgia Tech CyberGuide project [LKAA96] focuses on prototyping applications augmented with various positioning systems, potentially without communications at all. Using such an approach requires that devices be manually adapted to new environments.

Our conception of a "proxy server" is based on the model expressed explicitly in the Berkeley Client/Proxy/Server model [FBGA96] and implicitly in other work [BSAK95,AMZ95] that places application-level or network-level entities near, but not at, the endpoints of communications. This is another way of thinking about Active Networks [TSS+97], driven by end-to-end design principles [SRC84]: push agents to as close to the endpoints as possible, but no further. This concept of leveraging the well-connected, computationally powerful side of the wireless link (via "proxies" or "agents") pervades mobility research. It is also driven by the growing availability of workstation farms [APC+95] designed to provide compute resources for just such applications.

5.8 Continuing Work and Future Directions

Our continuing work involves iterating over the design, refining the implementation, and investigating various other approaches.

5.8.1 Wide-area issues

The current implementation has been tested only in a local area environment; work is continuing as to the specifics of how such servers aggregate (with union and intersection operations) and their hierarchy. This relates to naming issues and query semantics.

5.8.2 Building control and support

We are working with building architects and engineers at the Center for the Built Environment (http://www.ced.berkeley.edu/cedr/cbe) to incorporate devices such as the centralized heating and air conditioning, vents, fans, and temperature/humidity sensors into our system. This could allow users to close the environmental control loop and adapt areas in accord with user preferences as they move. Additionally, we hope to apply this model to the corporate environment team-based work process, allowing per-user location-based interfaces. For example, a R&D person visiting the Accounting division is probably interested in different services than the local workers in the same area.

5.8.3 Delegating operations

In general, mobiles may be allowed controlled access to CPU resources directly rather than configured services. This allows custom installation of “last-hop” network protocols, codecs, and security modules that are too compute-intensive to run on the end-client (e.g., for allowing the use of end-to-end session keys in an untrusted domain, for delta-encoding data, or for deploying private handoff prediction.) This requires a management layer for implementing policy decisions granting access to bandwidth, disk, and CPU. It also requires a mechanism for securely delegating operations [KWP97].

5.8.4 Queued RPC

Queued RPC mechanisms [ADT+95,BB97] support disconnection and link variability by incorporating application-managed messaging state. Queued RPC and asynchronous notification support is not incorporated into our system, but should be. (On the other hand, applications should also be able to ignore failed RPCs rather than queuing them, a more appropriate paradigm for situations such as with equipment interaction --- the client interface is designed to express the current state of external processes and most messages can specify idempotent operations.)

5.8.5 Maps

We wish to add additional functionality to our map application, including the ability to tie together physical geography to network connectivity. Servers could be pinned to their location on the floorplan and the connectivity graph automatically overlayed as it is discovered. Also to be added are the specifics of the administrative domains: overlaying the list of services available at groups of servers on the map, and extensions to illustrate hierarchy.

5.8.6 Interface specification grammar and compiler

A full specification of the ISL grammar and UI generation for different platforms is work-in-progress. Candidate base languages include HTML, Java, the CORBA IDL, a hybrid, or a fully customized design.

5.8.7 Fault tolerance

To maintain service on a local subnet in the face of a possible faults in the local SIP, we could leverage the property that beacons are equivalent to soft-state updates, and that therefore their disappearance implies a failure condition.

By having SIP servers maintain “neighbor” pointers to each other (administratively defined, or inferred through occasional expanding-ring broadcast searches), when SIP beaconing stops, these pointers to peer and/or hierarchical service beacon providers can be used to find another SIP still operating and accessible. Using a recovery protocol, an out-of-band request is made to extend the scope of the new SIP’s beacons to the affected subnetwork. This can be done by increasing the multicast TTL scope of the beacons to allow it to cross the transit network between the subnets or by using IP-in-IP encapsulation to tunnel. Such a solution addresses the failure condition by allowing access to remote services while the local fault is being corrected. The difficulty is determining whether maintaining/discovering these neighbor pointers is feasible.

5.8.8 Geographic locality

Currently there is no notion of requiring the tie between physical geography and network topology to be explicit. Users given IP access are expected to navigate through the global Internet where little or no locality is exposed even though it can be exploited. For example, the only hints of geographic information are out-of-band channels, heuristically through the IP interface domain name ("whitehouse.gov" in Washington, DC), IP address-to-city registration mappings available through *WhoIs*, or possibly the experimental DNS "LOC" record type.

Work has been done to allow clients to recreate these topological relationships for a small class of services using limited support from the network [GS94]. We'd like to overload our hierarchical service infrastructure this functionality directly. To do so, each service interaction server maintains "pointers" to others in the hierarchy and to peers (as in the above for fault tolerance). The pointers are then links in a geographic chain similar to the more familiar concept chains used in the WWW. In other words, just as HTML hyperlinks associate data based on content without regard to geography, neighbor links associate network topological locality without regard to content. Such links can be set up either manually with multi-lateral peering agreements (people agree to link topological neighbors), through occasional multicast expanding ring searches, or by inferring neighbors through the name and scope embedded in service advertisements. This requires no router support and can be incrementally deployed. As more links maintain a service advertisement beacon with these pointers, the leaves of the hierarchy would be filled out, allowing clients to infer a view of the geographic structure from their location. This gives us the possibility to enable a form of "window shopping" on the Internet, where neighbor networks can be queried to see what is available "next door."

5.8.9 Multimedia collaboration control architecture

The current suite of multimedia collaboration tools (vic, vat, wb, etc.) is focused on use at the desktop, with local control of each application through its user interface. In other words, the participant is also expected to be the controller.

These applications are now finding use in less traditional environments. One concrete example of this is the MASH Collaboration Laboratory, where media streams are sourced and sinked from a large number of non-computer devices. These devices (cameras, TV monitors, A/V routing controls, etc.) require remote (distributed) control to allow for the development of aggregate control applications that can configure such devices in combination.

We are developing a control infrastructure that can support such applications and developing prototypes for usability studies. The hope is to provide users with robust, intuitive room controls rather than requiring an attending technician to take care of such details. Additionally, the distributed control infrastructure will provide the mechanisms through which remote participants' applications can be controlled out-of-band (modulo policy-level access controls). Such mechanisms would relieve the need for users to receive control instructions (i.e., "Please turn down your source volume.") from technicians or advanced users through a sideband (or worse, in-band) channel.

5.8.10 Conference control primitives for lightweight sessions

The goal is to design a set of control mechanisms from which a wide variety of conference control (e.g., floor control) policies can be built. The base component is an announce/listen protocol to support "voting," much like SCUBA sender interest messages or RTCP receiver reports. Atop this message-passing framework are mechanisms for specifying the style of shared control (i.e., how votes are tabulated) for each element in the collaboration session. Also, we envision incorporating standard strong-crypto solutions for authentication and encryption to support access control lists and for assigning participants to "ownership classes" for the various objects in the environment.

A related open issue we are exploring is whether individual receivers and application-specific gateways should unify disparate announce/listen protocol messages. It seems that the "global, constant-bandwidth" allocations used by these protocols (i.e, SAP, RTCP, SCUBA) should not simply be summed as new protocols are deployed (a difficult predicament for low-bandwidth networks), but could instead share a single allocation. The hope would be to reduce the required announcement bandwidth by avoiding repetition of redundant data and to reduce consensus latencies by allowing individual protocols to adapt their share of a static allocation as necessary.

5.9 Conclusions

We propose that providing an "IP dial-tone" isn't enough. We must augment basic IP connectivity with adaptive network services that allow users to control and interact with their environment. The challenge is developing an open service architecture that allows heterogeneous client devices to discover what they can do in a new environment, and yet which makes minimal assumptions about standard interfaces and control protocols. We present our approach to implementing this, employing a few key techniques to realize this component of the architecture:

- augmenting standard mobility beacons with location information, scoping features, and announcements from a service discovery protocol;
- using interface specifications that combine an interface definition language with the semantics of a model-based user interface; and
- hosting scripts in the infrastructure that:
    - map exported object interfaces to client device control interfaces,
    - compose object interactions, and
    - automatically remap the destination of object invocations to changing server locations.

We also provide a detailed description of our prototype implementation of the service architecture and a number of example services in use at the UC Berkeley CS building.

6 The Network Stacks

In this section, we look at how the pieces fit together from the viewpoint of the network stacks. There are three primary stacks that matter: the basestation, client, and proxy stacks. The proxy really has two different kinds of stacks, those for clients and those for everything else such as distillers or servers.

Only the client-side stack differs from standard TCP/IP. We modify or extend the network stack at every level above the physical layer. In the next three subsections, we cover the three variations: at the client, at the basestation, and at the proxy. The network stack at the (legacy) server is by definition an unmodified TCP/IP stack. We will start with the basestation stack since it is the simplest and then cover the client and proxy stacks. Figure 22 shows the three variations of the network stack.

6.1 The Basestation Network Stack

There are two kinds of basestations: black-box basestations and Dædalus-aware basestations. The distinction is whether we can modify the code running on the basestation. For example, we expect to have no influence over cellular telephone basestations, so they're "black box"; but we have complete control over WaveLAN basestations, so they are Dædalus-aware. It is important to support both kinds so that we can improve performance when we have some influence, but still interoperate when we have no influence.

6.1.1 Black-Box Basestations

We would still like to optimize both network and application performance to the extent possible for black-box basestations. In the black-box case, the basestations is essentially invisible. It takes responsibility for routing, horizontal handoff among its the cells of its network, and whatever beaconing is required to find client devices.

For vertical handoff, our only control over a black-box basestation is the decision whether to use it at all. In particular, we use the basestation exactly when we use that particular network. In such cases, the network typically assigns a particular IP address and we must use it if and only if we use this network. Thus, vertical handoff is equivalent to switching the IP address of the client.

6.1.2 Dædalus-Aware Basestations

Given some control over the basestations, we can improve both the network performance and the quality of vertical and horizontal handoffs.

Figure 22a shows the layers of the network stack at a Dædalus-aware basestation. Typically, the basestation will have two physical networks: an ethernet to connect to the internet and the physical network for which it is a basestation, such as WaveLAN.

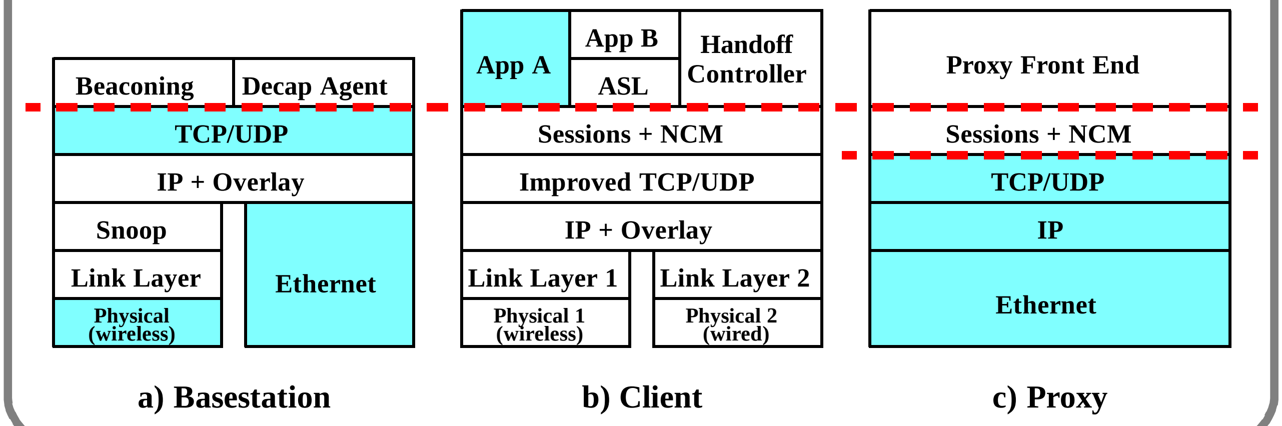


FIGURE 22. The three variations of network stacks: the basestation (a), the client (b) and the proxy (c). Shaded regions indicated unmodified layers and the dashed line indicates the likely location of the user/kernel boundary. Thus, unshaded regions below this line imply kernel modifications. The "Sessions + NCM" layer can go either into the kernel or a user-level library. Note that a client can run both modified and unmodified apps simultaneously.

The following table summarizes each of the modules and our modifications.

TABLE 2. Network stack on the basestation.

Module	Description	Dædalus Modifications
Physical	Low-level hardware and software for the (wireless) network.	None
Link	Management of the shared media for those networks that share (nearly all wireless networks).	Link scheduling for wireless LANs (see Section 3.3) • Admission control and bandwidth allocation for MHs • Link usage tracking and distribution of surplus bandwidth
Snoop	Monitors TCP traffic for packet loss due to errors (not congestion).	Snoop Protocol (see Section 3.1): • Local retransmission of packets lost due to wireless errors
IP + Overlay	Standard IP Routing + Mobile IP + IP Multicast	• State for each client: forwarding packets, buffering packets, or forwarding only IP headers • Fast horizontal handoff (based on multicast preloading of neighboring cells) • Support for co-existence with standard IEFT Mobile IP at client • See Section 2

TABLE 2. Network stack on the basestation.

Module	Description	Dædalus Modifications
TCP/UDP	Standard TCP/UDP. Used only to communicate with the basestation, rather than affecting client traffic.	None
Beaconing	Broadcasts network beacons for this cell. Clients detect these beacons and can then connect to the basestation.	• Notification of wired IP address • Name server location, See Section 5.2 • Possible piggybacked data, See Section 5.4
Decap Agent	Communicates with Handoff Controller on the client and controls the routing state and Foreign/Home agent mapping in the IP + Overlay layer.	• TCP communication with the client • Control over Overlay routing

6.2 The Client Network Stack

The client stack is also highly modified, and adds several new pieces to the stack from the basestation:

TABLE 3. Network stack on the client.

Module	Description	Dædalus Modifications
Physical Layer	Low-level hardware and software for the (wireless) network.	• API for determining whether network interface is on/off and for measuring current signal quality
Link Layer	Management of the shared media for those networks that share (nearly all wireless networks).	Link scheduling for wireless LANs • Enforcement of bandwidth allocation
IP + Overlay	Standard IP Routing, Mobile IP, and IP Multicast	• Track performance of multiple interfaces • Signal higher layers when they should re-examine the choice of network • Support for co-existence with standard IEFT Mobile IP in infrastructure
TCP/UDP	TCP/UDP with modifications for improved performance on wireless and asymmetric networks.	• Selective acknowledgments • Asymmetric network enhancements • See Section 3

TABLE 3. Network stack on the client.

Module	Description	Dædalus Modifications
Sessions + NCM	Multiplex many short connections (such as HTTP) onto one long-lived TCP connection, or "session". Provide network measurement statistics on a per session basis.	• TCP multiplexing onto one connection • Network monitoring for each session • Signal user for significant changes in network performance • See Section 3.4
Application Support Layer (ASL)	Provide fine-grain application-level control over the proxy. Examples include control over distillation and refinement.	• See XXXX
Handoff Controller	Monitor each of the physical networks using their new API, control which network to use.	• User-level process • Directly monitors each network interface • See Section 2.2

6.3 The Proxy Network Stack (for Clients)

At the proxy, the network stacks are much simpler. The wired side (used to communicate with servers) is an unmodified stack that typically uses Ethernet. The connection to the base station also uses Ethernet, but uses the Sessions and TCP protocol enhancements for better performance and network monitoring.

The following table describes the interesting network layers on the proxy:

TABLE 4. Network stack used by the proxy to communicate with clients.

Module	Description	Dædalus Modifications
Sessions + NCM	Maintains long-lived session with the client, containing many short connections (typically HTTP). Provides network performance information to the proxy front end.	• TCP multiplexing onto one connection • Network monitoring for each session • Signal proxy front end for significant changes in network performance • See Section 3.4
Proxy Front End	Interface to proxy for all of the traffic to one client. Single flow of data enables scheduling and control of precious proxy-client bandwidth.	• Controls distillation based on current client and network parameters

6.4 Network Stack Summary

These stacks contain extensive modifications to traditional TCP/IP and higher layers. However, each modification adds value and the changes are broken into discrete orthogonal sections: TCP performance (Snoop protocol, selective acknowledgments and enhanced loss recovery, sessions, network connection monitoring) Overlay Networking, IETF Mobile IP compatibility, and proxy and application support. The network services modules (authentication, name server, etc.) have no effect on the stacks, although the network services module affects the contents of beacons.

7 Summary

In this section, we revisit the original ten principles and summarize how each principle is met by the architecture. By implementing these principles, we also meet the larger goal: best-quality multimedia access anytime, anywhere with heterogeneous networks and clients in an infrastructure that scales, remains available, and is easy to use.

7.1 The Principles Revisited

Principle 1: ***Heterogeneous Networks****: The infrastructure must include wireless networks that have a mixture of global and indoor coverage, thus requiring a heterogeneous collection of networks.*

This principle is supported by a combination of Overlay Networking, TCP performance and the proxy. Overlay networking provides detection of the available networks and manages handoffs among them, both vertically and horizontally. The TCP improvements mitigates problems with many of the wireless networks, such as high error rates and asymmetry, and the proxy tunes the content to the available bandwidth, thus widening the range of practically useful networks.

Principle 2: ***Scalable****: The infrastructure must scale to support millions of users.*

We assume that the physical network is scalable; this is true in practice simply because there is no need for centralization. Similarly, the network services system is scalable because it is replicated locally. The challenge is to scale the proxy, which is a shared resource so that it can be amortized over many users. The proxy has been redesigned for scalability using a NOW cluster and extensive multithreading. It also exploits burstiness is user needs to increase the aggregate number of supported clients.

Principle 3: ***Highly Available****: The infrastructure must be available all of the time.*

We mostly assume the availability of individual networks such as Metricom. However, the overlay networking concept increases overall availability dramatically by offering multiple networking options. Thus the loss of an individual network need not imply loss of service. In addition, the proxy and network services systems provide highly available service through replication, automatic failover, and fast restart of subsystems that fail. For example, the PTM has a primary and secondary and distillers will restart the PTM if they detect that it has failed.

Principle 4: ***Transparent Access****: The detection and setup of a network connection should be automatic. Users shouldn't have to know what networks are in range.*

The principle is implemented by overlay networking and by the network services system. In particular, overlay network handles the detection of networks and automatic connection to them, while the services module provides automated location of common critical services such as DNS, e-mail, and proxies. Thus users get access to all of these resources (and many others) simply by being in range of a Dædalus-aware network.

Principle 5: ***Localized Service****: The detection and setup of local network services should be automatic. Users shouldn't have to know what services are available at their current location.*

This principle is the driving force behind the network services system. It handles the detection of available services, automatic connection to critical services (like DNS), and even dynamic extension of the client with new (local) abilities though code mobility.

Principle 6: ***Global Authentication****: We must authenticate users using a globally available security infrastructure, such as public-key cryptography or Kerberos.*

Authentication is handled mostly by Kerberos, but we have extended Kerberos to impoverished clients via Charon, which is part of the network services module. This allows reliable end-to-end authentication that exploits the infrastructure without having to trust it. We use Kerberos' cross-realm authentication protocol to handle mobile hosts that appear in a foreign (untrusted) environment.

Principle 7: ***Multimedia****: The infrastructure must support graphics, audio and video in addition to text.*

Multimedia support comes from the proxy, the optimized transport layer, and even overlay networking. The proxy tunes multimedia to the current format and bandwidth requirements, thus enabling multimedia access in situations where it was previously impossible. The use of delivery classes allows the network to understand the transmission requirement of multimedia data and optimize and prioritize accordingly. Finally, the overlay system provides fast handoff by precharging neighboring cells, which enables smooth audio and video across handoffs.

Principle 8: ***Performance****: The user's data should arrive as fast as possible. This includes selecting the best network, optimizing the network performance, and optimizing the content at the application level.*

The optimal performance for the current client device and location comes from the combination of the TCP optimizations, the overlay networking module, and the proxy. TCP optimizations mitigate wireless errors, improve throughput with long-lived sessions and support for asymmetric networks, and improve utilization through delivery classes and scheduling for shared links. The overlay module helps by selecting the best network for the current location. Finally, the proxy greatly improves performance (especially latency) by tuning the content for the current network conditions, which it knows through network monitoring of the session with each client.

Principle 9: ***Heterogeneous Client****s: Complexity should be pushed into the infrastructure, where it can be amortized over all of the active users. The infrastructure should support both inexpensive client devices, such as smart phones, and more sophisticated computers, such as high-end laptops.*

This is the driving force behind the proxy architecture. The only way handle both widely heterogeneous clients and legacy servers is to adapt content dynamically for each client. The scalable proxy solves this problem efficiently through datatype-specific distillation and refinement, and through a scalable extendable architecture.

Principle 10: ***Dynamic Adaptation****: The data sent to the user should be optimized for timeliness, carrying the most information in the least amount of time. The nature of this adaptation depends on the current network, the preferences of the user, and the nature of the data (text is much different than graphics).*

This principle is also tightly tied to the proxy. The proxy knows current network conditions, tracks user preferences, caches object before and after transformation, and knows the nature of the data because of the use of MIME types and datatype-specific distillation. This information, combined with fast (real-time) transformation engines, allows the proxy to customize content completely for each client individually on every access.

7.2 Summary

Finally, we return to our original high-level goal:

> *"People and their machines should be able to* **access** *information and communicate with each other* ***easily*** *and* ***securely****, in* ***any medium*** or combination of media*— voice, data, image, video, or multimedia —* ***any time****,* ***anywhere****, in a* ***timely****,* ***cost-effective*** *way."*
>
> *Dr. George H. Heilmeier*
> *IEEE Communication*
> *October 1992*

How far are we away from this goal given our architecture? We believe that we have provided a solution that handles the easy and secure multimedia access aspects, and provides timely cost-effective service. The remaining issue is that of "any time, anywhere." Assuming the wide deployment of wireless networks, which is a big assumption, we believe this architecture meets these goals as well. In particular, overlay networking and the scalable, highly available infrastructure allow clients to use any available network to obtain best-quality network access any time, anywhere (supported by at least one network).

8 Glossary

ASL See *Application Support Layer*, XXXX

BARWAN *Bay Area Research Wireless Access Network*: The testbed used in our project to interoperate multiple wireless networks. The testbed currently includes WaveLAN, IR, Metricom, Hughes DBS, CPDP, and cable modems.

Basestation The transmitter and router for a single cell of a wireless network. Typically, the BS is connected via ethernet to the infrastructure components and provides beaconing, forwarding, and some measurement and control over the cell.

Black-Box BS A Basestation that we have no control over in terms of modifying code running on the basestation. Examples are CDPD and Metricom.

Beacon Agent This agent sends out periodic beacon packets that are similar to Mobile IP agent advertisements but may also contain additional information

BS See *Basestation*

CB See *Coordination Bus*

COA See *Care-Of Address*

Care-Of Address The visited locale's temporary address for a mobile host, advertised to the home agent (and to corresponding hosts using route optimization)

Cell A not necessarily contiguous physical area covered by a single transmitter in a wireless network. Cells may overlap with both other cells of the same network and with cells of other wireless networks.

Channel A multicast group with a symbolic name that can be looked up via the name server. It is the unit of grouping in the coordination bus.

Coordination Bus This is the shared communication medium among the modules that collectively form the proxy. The bus consists of a number of named channels to which any module can broadcast or listen. Channels map one-to-one with multicast IP addresses and the name server maps channel names to these addresses.

Dædalus The networking half of this project, which includes all of the TCP and handoff stuff.

DBS *Direct Broadcast Satellite*

Decapsulation Agent This agent receives messages from the Handoff Controller and modifies Overlap IP's kernel-level translation tables in response to these messages.

DHCP *Dynamic Host Configuration Protocol*. This protocol can be used to assign a local IP address to a device that just joined the local network.

FA See *Foreign Agent*

Foreign Agent This entity resides in "visited" systems, advertises its presence, and is used to obtain a temporary care-of address (either multicast or unicast) for a mobile host.

GloMop
The application services half of this project, which includes the proxy and security aspects.

HA
See *Home Agent*

Handoff
The change from one network connection to another. Ideally, this transfer is transparent to the application. If done at the link layer, which is common for horizontal handoffs, it is transparent even to TCP.

Handoff Controller
This component handles the transition when a mobile device moves from one base station to another (horizontal handoff) or when a mobile device moves from one network to another (vertical handoff). It sends the encapsulation/decapsulation requests to the decapsulation agent. It makes all handoff decisions due to mobility by listening to beacon packets sent by the beaconing agent.

Home Agent
As in Mobile IP: A router on a mobile node's home network which tunnels datagrams for delivery to the mobile node to its Care-Of Address when it is away from home, and maintains current location information for the mobile node.

Horizontal Handoff
A handoff between cells within the same physical network, such as from one Metricom pole-top basestation to another.

Metricom
Makers of the Ricochet packet radio modem. The Metricom network is a half-duplex store-and-forward network that is deployed in the Bay Area, Portland, and Washington, D.C. The bandwidth is about 30 kbps, but the latency can be seconds.

MH
See *Mobile Host*

MN
See *Mobile Node*

Mobile Host
This is the client device; See*Mobile Node*

Mobile Node
This terminology comes from Mobile IP: A host or router that changes its point of attachment from one network or subnetwork to another. A mobile node may change its location without changing its IP address; it may continue to communicate with other Internet nodes at any location using its (constant) IP address, assuming link-layer connectivity to a point of attachment is available.

Mobile IP
An extension to IP that supports mobility by forwarding packets through a home agent, which tracks the location of the mobile host.

NCM
Network Connection Monitor: Part of the sessions functionality that provides feedback on the current state of the session. Statistics collected include the packet loss rate and effective bandwidth of the transport-level connection.

Overlay IP
Extensions to IP that support mobility and handoff across different networks.

Proxy
A well-connected node that acts as an intermediary and translator between poorly connected, heteregeneous clients and the rest of the internet.

PTM
Proxy Transcoder Manager: The PTM manages distillers for the proxy front ends. It controls a pool of distillers for each data type, provides load balancing, and handles spawning and killing distillers as the overall load varies over time.

Ricochet	See *Metricom*
Session	A long-lived connection over which many short TCP connections are multiplexed. It also provides connection statistics using the NCM.
User Control Panel	This allows user-specific customization about the choice of network or base station to connect to as well as the choice of handoff policy to use (e.g. aggressive low-latency handoff vs. less aggressive high-latency handoff).
Vertical Handoff	A handoff between different networks, such as from IR to WaveLAN.
VGW	See *Video Gateway*
Video Gateway	A distiller for real-time video traffic.
WaveLAN	A radio-based local-area wireless network.

9 References

[ABSK95] E. Amir, H. Balakrishnan, S. Seshan, and R. H. Katz. Efficient TCP over Networks with Wireless Links. *Proc. Fifth Workshop on Hot Topics in Operating Systems (HotOS-V),* Orcas Island, WA, May 1995.

[AMZ95] E. Amir, S. McCanne and H. Zhang. An Application Level Video Gateway. In *Proc. ACM Multimedia '95*, San Francisco, CA, November 1995.

[Ardis] Ardis web page. http://www.ardis.com, 1996.

[BDH+94] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz, and D. P. Wessels. *Harvest: A Scalable, Customizable Discovery and Access System*. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Boulder, August 1994 (revised March 1995). ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.Jour.ps.Z.

[BSAK95] H. Balakrishnan, S. Seshan, E. Amir, R. H. Katz. Improving TCP/IP Performance over Wireless Networks. *Proc. 1st ACM Conf. on Mobile Computing and Networking*, Berkeley, CA, November 1995. Best Student Paper Award.

[BSK95] H. Balakrishnan, S. Seshan, R. H. Katz. Improving TCP/IP Reliable Transport and Handoff Performance in Cellular Wireless Networks. *ACM WirelessNetworks,* 1(4), December 1995.

[Cac95] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications*, 13(5), June 1995.

[CDPD] V. Garg and J. Wilkes. *Wireless and Personal Communications Systems*. Prentice-Hall, 1996. Chapter 8.

[DN95] Antonio DeSimone, Sanjiv Nanda. Wireless Data: Systems, Standards, Services. In *Wireless Networks* 1 (1995) 241-253.

[FJ95] Sally Floyd, Van Jacobson. Link-Sharing and Resource Management Models for Packet Networks. In *IEEE/ACM Transactions on Networking*, Vol 3, No 4, August 1995.

[FG96] A. Fox, S. D. Gribble. Security On the Move: Indirect Authentication Using Kerberos. *Proc. 2nd ACM Conference on Mobile Computing and Networking (MOBICOM 96)*, Rye, NY, October 1996.

[FGBA96] A. Fox, S. D. Gribble, E. A. Brewer, E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. *Proc. Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII),* Cambridge, MA, May 1996.

[HH94] Andy Harter and Andy Hopper. A Distributed Location System for the Active Office. *IEEE Network Magazine*, 8(1), January 1994.

[IDM91] J. Ioannidis, D. Duchamp, and G. Maguire. IP-Based Protocols for Mobile Internetworking. In *ACM SIGCOMM Symposium on Communications, Architecture, and Protocols*, pages 235-245, 1991.

[LM96] G. Liu and G. Maguire, Jr., A Class of Mobile Prediction Algorithms for Wireless Mobile Computing and Communications. In *Journal on Special Topics in Mobile Networks and Applications*, Volume 1(1996), No. 2.

[Jac88] V. Jacobson. Congestion avoidance and control. In *Proc. of SIGCOMM '88*, August 1988.

[Lee94] J. s. Lee. Overview of the technical basis of qualcomm's cdma cellular telephone system design: a view of north american tia/eia is-95. In *Proc ICCS '94*, volume 2, pages 353–358, Nov 1994.

[Per96] C. Perkins. *IP Mobility Support*. RFC-2002, October 1996.

[PM94] V. N. Padmanabhan and J. C. Mogul. Improving HTTP Latency. In*Proc. Second International World Wide Web Conference,* October 1994.

[Ricochet] Metricom web page. http://www.metricom.com, 1996.

[SAG+93] Bill N. Schilit, Norman Adams, Rich Gold, Michael Tso, and Roy Want. The PARCTAB Mobile Computing System. In *Proceedings Fourth Workshop on Workstation Operating Systems (WWOS-IV)*, pages 34-39. IEEE, October 1993.

[Ses95] S. Seshan. *Low Latency Handoffs in Cellular Data Networks*. Ph.D. thesis, University of California at Berkeley, December 1995.

[Ste88] J. G. Steiner, C. Neuman, and J. I. Schiller. Kerberos: an authentication service for open network systems. In *Proceedings of the USENIX Winter Conference 1988*, pages 191–202, Dallas, Texas, USA, February 1988.

[Ste96] M. Stemm. *Vertical Handoffs in Wireless Overlay Networks*. M.S. Report, University of California at Berkeley, May 1996.

[Ste94] W. R. Stevens. *The TCP/IP Illustrated, Volume 1*. Addison-Wesley, Reading, MA, Nov 1994.

[Ste97] M. Stemm and R. H. Katz Vertical Handoffs in Wireless Overlay Networks*ACM Mobile Networking, Special Issue on Mobile Networking in the Ineternet*, Fall 1997.

[Tek91] Tekinay S. and B. Jabbari. Handover and Channel Assignment in Mobile Cellular Networks. *IEEE Communications Magazine*, 29(11):42–46, November 1991.

[Zha86] L. Zhang. Why TCP Timers Don’t Work Well. In*Proceedings of ACM SIGCOMM ’86*, pages 397–405, August 1986.