Dynamic Network Reconfiguration Support for Mobile Computers

Jon Inouye
jinouye@cse.ogi.edu
Oregon Graduate Institute
Beaverton, Oregon

Jim Binkley
jrb@cs.pdx.edu
Portland State University
Portland, Oregon

Jonathan Walpole
walpole@cse.ogi.edu
Oregon Graduate Institute
Beaverton, Oregon

Abstract

Hot swapping technology combined with pervasive heterogeneous networks empowers mobile laptop users to select the best network device for their current environment. Unfortunately, the majority of system software remains "customized" for a particular network configuration, and assumes that many characteristics associated with the network environment remain invariant over the runtime of the software. Mobility causes changes in the environment and nullifies many of these assumptions. This leads to severe loss in system functionality when resources are lost, and failure to benefit when resources are gained.

This paper describes Physical Media Independence (PMI), an architecture for dynamically diverse network interface management. PMI addresses three issues concerning dynamic network configuration. First, a model for device availability is proposed to accurately determine when a network device is operational. Second, a structured methodology is used to construct adapters that reconfigure the system when the set of available devices change. The methodology breaks traditional layering using a meta-interface to pass events and information among layers, allowing each layer to adapt in a manner best suited to it. Third, we examine the effect of transparent and non-transparent reconfiguration operations on a variety of applications. We find that adaptive, resource intensive applications behave more efficiently when informed of device availability events. We compare the functionality of an adaptive application running on top of a adaptive operating system (OS) with and without cross-layer notifications.

1 Introduction

Large variations in both hardware configurations and network environments are made possible by advances in interface technology and rapid deployment of network infrastructure. "Hot swappable" interfaces such as PCMCIA [18] and IEEE 1394 [30] provide users with the ability to dynamically attach or remove devices while the system is running. There is no need, other than software limitations, to power cycle the machine, reboot the operating system, or restart applications while reconfiguring the system.

Unfortunately, the richness of the environment poses an interesting network management question: "How can a mobile host be reconfigured transparently as it migrates across different network environments and uses heterogeneous network interfaces?" Due to the static nature of their environment, desktop systems typically hard-code network configuration information in files. Extending this approach to mobile computers implies a configuration file for every possible combination of interfaces and environments. While this approach may suffice for a limited number of well-known networks, e.g., home and office, the complexity grows with the number of environments. Emerging dynamic configuration technology is helping to address this issue but, as will be shown, tends to focus on hosts equipped with a single network interface.

The goal of Physical Media Independence is *desktop equivalence for network management.* By desktop equivalence, we imply that a mobile computer should not be significantly harder to configure than a stationary computer. To achieve this goal, the network configuration must adapt itself as active interfaces become disabled and new interfaces become available. The methodology we use is based on the concepts of *explicit assumptions* and *intelligent adaptation*.

- *Explicit assumptions.* Once a device is available, we assume that it continues to remain available, rather than running a series of checks on the device before sending each packet to it. Explicitly checking for device availability is expensive, so instead, we make the optimistic assumption that the device will remain available, and move the checks from the frequently used packet-processing path to more rural device management paths that are traversed rarely. We call these checks *guards* since they monitor the validity of our assumptions. We describe a device availability model that defines the set of characteristics that identify an available device. Each characteristic is guarded by the addition of code to detect changes in that characteristic.
- *Intelligent adaptation.* Guards trigger adaptation policies when they detect changes in the environmental features they monitor. Ideally, adaptation is intelligent with adaptation policies in different layers cooperating toward a common goal. A particular layer should not try to perform all the necessary adaptation by itself because it may duplicate or interfere with adaptations better performed by other layers.

The remainder of this paper is organized as follows. Section 2 presents our model supporting device availability. The model answers the question, "What devices are available?" Section 3 describes our approach to intelligent adaptation and describes how the system reconfigures itself when the set of available devices changes. Section 4 describes our implementation of the system and Section 5 presents an evaluation of the implementation. Section 6 describes the effects of our adaptation approach on various types of application programs and Section 7 discusses the implications. Section 8 follows with a description of related research, and we summarize our experiences in Section 9.

To appear in MobiCom'97, Budapest, Hungary, September 26-30, 1997.

Table 1: Device Characteristics

Name	Description
Present	Device is physically attached
Connected	Link-level connectivity
NetNamed	Network name is bound to device
Powered	Power is available
Affordable	Cost of use is within budget
Enabled	User choice

2 Device Availability Model

We designate network interface availability as the primary assumption in providing Physical Media Independence. Interfaces can be *available* or *unavailable* to higher level protocols. When an interface is available, packets sent to it by the network layer are passed to the link layer where they are encapsulated into frames and sent out via the physical network device. Frames received by the physical network device are passed by the device driver up to the network layer. Packets do not flow between the network stack and the device driver when the driver's network interface is unavailable. We define six characteristics that determine when an interface is available. These characteristics are shown in Table 1. We concede the likely existence of other useful characteristics, such as security, but believe they can be integrated into our model without changing the overall architecture.

An interface is *present* if both the hardware device and the software driver exists. This characteristic addresses the issues emerging with hot swapping technology.

An interface is *connected* if there exists a connection at the link layer.[1] This characteristic addresses the variable connectivity of wireless networks. Fluctuations caused by distance, interference, and obstructions may prevent a wireless interface from communicating with its base station. While connectivity is not a boolean value, our model requires a simple event to be raised to mark the device unavailable or available. The policy behind the event notification may be multi-valued. For example, a simple empirically determined threshold value for signal strength might be used, or a more complex feedback loop involving higher level protocols. Wired interfaces also require link connectivity checks due to cabling. Users may disconnect cables and leave PCMCIA network cards attached to their computers while mobile because it is more convenient than removing and storing the card.

An interface is *network named* when there exists a binding to a network address. This characteristic has been required for supporting network (and higher) level protocols. We concentrate on IP networks, so we require an IP address assignment to satisfy this condition.

An interface is *powered* if the network device has enough power to function properly. The interface does not have to be powered for this characteristic to be true. For example, power conserving "sleep" mode may remove power from sections of a network device while not in use, and restore power when those sections are needed. The *powered* characteristic concentrates on situations such as suspend/resume operation, detecting a failing external battery, or interacting with operating system power management policies.

An interface is *affordable* if the monetary cost of using the device is within the budget. Because wide-area wireless services have limited bandwidth, providers tend to charge on a per-usage basis. Examples include both circuit cellular (GSM, AMPS), which charges users per unit time, and packet cellular (CDPD), where charges are based on the volume of data traffic. While beneficial for interactive users who don't realize how long they are "surfing" the World Wide Web or how much data they are transferring, this characteristic targets *virtually connected* systems. Virtual connectivity is support by PPP clients that automatically establish a point-to-point connection when a packet is sent to the interface, and "wake-on-ring" technology that resumes a suspended computer when an incoming call is detected. When virtual connectivity is used by *background* traffic, this characteristic provides a "cutoff" switch to prevent unexpectedly large bills.

[1] So a twisted pair Ethernet interface card connected to an isolated hub would be considered "connected".

Table 2: Internal Events Generated by External Events

External Events	Internal events
Card removed	Invalidate name and connectivity, power down.
Card attached	Power up, check connectivity, begin name acquisition.
Power suspend	Invalidate name and connectivity.
Power resume	Check connectivity, begin name acquisition.
Connectivity loss	None.
Connectivity gain	None.
Name acquisition	None.
Name invalidation	Begin name acquisition.
Affordable	Power up, check connectivity, begin name acquisition.
Unaffordable	Invalidate name and connectivity, power down.
User disable	Invalidate name and connectivity, power down.
User enable	Power up, check connectivity, begin name acquisition.

An interface is *enabled* unless it is specifically disabled by a user-level directive. This provides users with the ability to override the system and disable a device without having to remove or disconnect it. UNIX systems have supported this characteristic as a flag within an interface's data structure representation.

An interface is *available* if the conjunction of all these characteristics is true (Equation 1). Once we determine whether a device is available or unavailable, we can configure the system based on the set of available devices. This configuration is valid until the membership of the set changes.

$$Available(I) \vdash I_{present} \wedge I_{powered} \wedge I_{connected} \wedge I_{netnamed} \wedge I_{affordable} \wedge I_{enabled} \quad (1)$$

Given all these characteristics, each network interface may be represented by a state machine. Each state represents a combination of the truth values of the six characteristics. Different state machines are possible by varying the policies driving state transitions. For example, in our state machine all LAN devices start out with the *enabled* characteristic while point-to-point interfaces start without it. A state machine is affected by both external events (user attaching a network card) and internal events generated in response to the external events. Table 2 describes one possible set of internal state transitions caused by external events. There are many tradeoffs. For example, our state machine invalidates the network name characteristic when suspending, disabling, or removing a network device. This forces name acquisition to be performed before the device can be made available again. Optimistically, one could assume the previous network name would be valid until proven otherwise. The goal is faster reconfiguration performance and if the assumption is correct most of the time, then performance will improve. If the assumption is incorrect, the system must first detect it and then recover. Another tradeoff is power consumption. A more power-conservative state machine might power down an interface and disable it upon losing link connectivity rather that to continue to power it and hope connectivity will be restored soon. This would force the user to enable the device (or reboot the system) before the device could be operational again.

2.1 Detecting Invalidations

We place *guards* to detect the validation and invalidation of device characteristics. These guards are placed in modules that monitor the validity of the characteristic and are not present within the control path used to send or receive packets. When triggered, guards force transitions in the state machine and may create a different set of available devices. In this section, we focus on the mechanisms needed to monitor our device characteristics.

Guarding *Present*. We depend on the existence of hot swapping support within the operating system, such as Windows Plug-and-Play [10], or hardware interfaces, such as those provided provided by PCMCIA[18].

Guarding *Connected*. The most logical place to guard this property is the device driver. More advanced driver interfaces are beginning to support this type of functionality. For example, the wireless NDIS[2] extensions proposed by the Personal Computer and Communications Association (PCCA) allow network stacks to receive indications when channel quality falls below a specified threshold [23]. Unfortunately, most driver interfaces for wired devices do not export link-layer connectivity to the operating system even though the hardware supports it. The IEEE 802.3 10BASE-T standard defines a twisted-pair Ethernet link integrity test but does not require all devices to implement it. Those that do have no standard programming interface to communicate this information to higher layers and usually end up logging warnings to the system console.

Guarding *NetNamed*. Network names are invalidated when migrating to a new network. There are many mechanisms for discovering this including receiving ICMP router advertisements [8], Mobile IP beacons [22], querying DHCP [9] servers, or receiving user input. These guards will most likely live within network-layer entities and must propagate their events down to the device manager. From an opposite viewpoint, the device manager must be able to upcall network-layer entities to acquire a network name.

Guarding *Powered*. Power is becoming an increasingly important resource. We assume the existence of some form of power management module that is responsible for allocating and deallocating power resources in a system. While APM [12] provides a few basic mechanisms for power management, it does not provide any power management policies or ability to monitor the power usage of network devices. Second-generation system like the Advanced Configuration and Power Interface (ACPI) [3] are designed to give the operating system more control over power management responsibilities.

Guarding *Affordable*. Determining whether or not an interface is within a budget depends on knowing both the budget and the cost of using the interface. Our architecture places the burden of providing this information on the user or some other higher-level entity. The user is responsible for setting a strict budget for network communication and binding a "cost model" to all interfaces charging usage fees. Given these two pieces of information, the system is responsible for monitoring the interface and disabling it when the budget is exceeded.

[2] Network Driver Interface Standard – the driver interface for Microsoft Windows.
[3] See http://www.teleport.com/∼acpi/ for more information on ACPI.

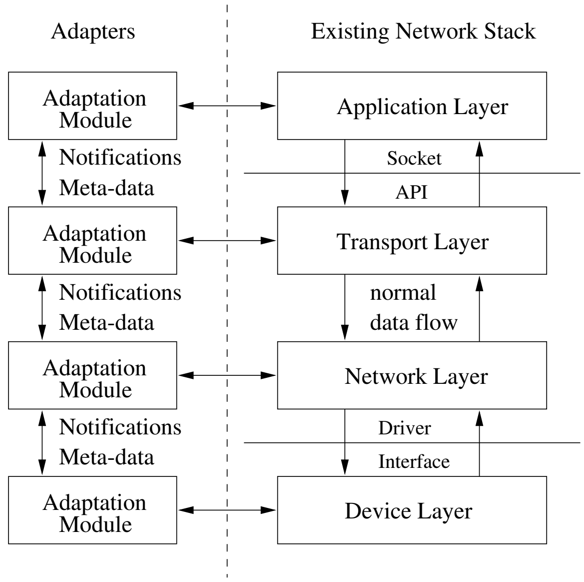

Figure 1: Layered Design w/ Cross-Layer Notification

Guarding *Enabled*. User directives can be guarded by adding a check to the application programming interface (API) used to enable or disable the device. Most operating systems provide such an API, and guarding it should only require the addition of an event mechanism that triggers when the API is used to toggle this characteristic.

3 Intelligent Adaptation

We say a system adapts intelligently if it avoids conflicts and duplication of effort between layers. This goal is accomplished by communicating assumptions and adaptation operations across layer boundaries. Figure 1 illustrates the application of this approach to a typical TCP/IP network stack. Adaptation modules attached to each layer of the network stack receive policy-related information (meta-data) from higher layers and event indications from lower layers. We call these types of messages *cross-layer notifications*.

Changes in device availability are associated with the link layer, but cross-layer notifications are used to propagate this information to higher layers. In our model, a layer delays propagating device availability indications to higher layers until its own internal adaptation has completed. While this is helpful in avoiding synchronization problems, it increases the latency in propagating notifications.

3.1 Network-Layer Adaptation

The IP network layer creates two forms of bindings to interfaces: multicast groups and route table entries.[4] IP multicast groups are bound to interfaces, allowing interfaces to participate in group management (IGMP) communication between host interfaces and multicast routers [7]. If an interface is removed, the system must decide what to do with the groups attached to that interface. Should group membership be revoked, or should they be migrated to another available interface? Revoking group memberships prevents

[4] In 4.4BSD ARP entries are route table entries.

applications from receiving packets sent to that group while migrating group memberships may adversely affect the network. For example, you would not want to migrate a group with high bandwidth requirements to an interface that could not support it.

Route table entries bind destination addresses to interfaces. When interfaces become unavailable, these entries must be invalidated. When a new interface becomes available, bindings discovered via routing protocol advertisements and user-specified entries must be created.

3.2 Transport-Layer Adaptation

The transport layer caches a lot of information. Protocol control blocks cache route table entries to avoid route lookups on every packet. In 4.4BSD, route table entries store characteristics such as maximum transmission unit (MTU), slow start threshold, and estimated round-trip-time (RTT) associated with a route [31]. When network layer adaptation migrate routes from one network device to another (or from one network address to another), notifications should trigger recalculations of these characteristics.

3.3 Application-Layer Adaptation

Applications also have assumptions based on available network devices and cache information related to these devices. The most prominent of these is the IP address of the local interface which is stored in the socket structure. However, applications also care about path characteristics such as available bandwidth, connectivity, trust, and cost.

4 Implementation

We implemented our system using FreeBSD 2.1R augmented by the April 1996 release of the PAO patch kit and Portland State University's Mobile IP [22] release.[5] We chose FreeBSD due to the freely distributed nature of the code, low cost, commercial quality of the network stack, and the available documentation. The disadvantages of choosing FreeBSD are the alpha nature of the PC card and APM [12] support. The FreeBSD network code is derived from 4.4BSD [17] and is documented thoroughly in [31].

Figure 2 illustrates the various components of our prototype system. The prototype is very modular in nature and is implemented mostly outside the kernel. The PC card daemon, **pccardd**, is included with FreeBSD and supports hot swappable PCMCIA devices. **pmid** implements multiple device availability state machines and the policies used in supporting Physical Media Independence. On startup, **pmid** reads a configuration file (pmid.conf) containing configuration policies and administrator-specified network information. The policies specify preferences that help lower layers adapt in a manner providing the greatest user benefit. For example, if multiple possible default routers exist on multiple interfaces, one policy helps determine which interface to use. Once the configuration file has been assimilated, **pmid** examines each interface on the system and removes those interfaces it knows are not associated with network devices, such as the loopback interface. After initialization, **pmid** listens for messages from the guards and periodically queries the kernel for network interface status to make sure its representation of network interfaces remains consistent with the kernel. This high-level guard helps catch events that slip past other guards in the system. A pseudo device driver, /dev/pmi, is used to pass events from the operating system to **pmid** and provides a meta-interface for applications to register interest in, and receive notifications of, device availability events. The mobile node daemon, **mnd**, is part of PSU's Mobile IP distribution and handles handoffs for a single class of interface cards. Unfortunately, the integration of Mobile IP with our dynamic reconfiguration daemons is not complete and there are some outstanding issues, such as routing protocol interaction, that still need to be addressed.

[5] See http://www.jp.FreeBSD.org/PAO for more information on the PAO patches and http://www.cs.pdx.edu/research/SMN for information on PSU's Mobile IP release.

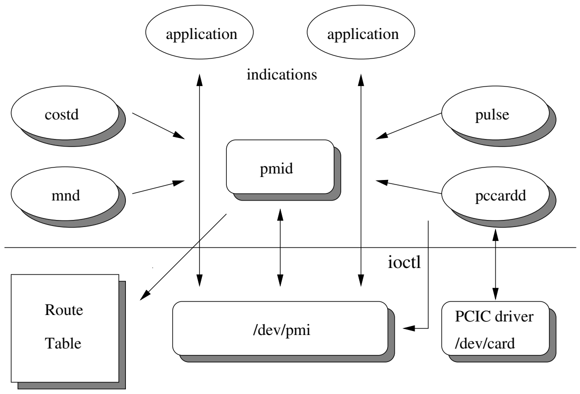

Figure 2: Dynamic Configuration Support

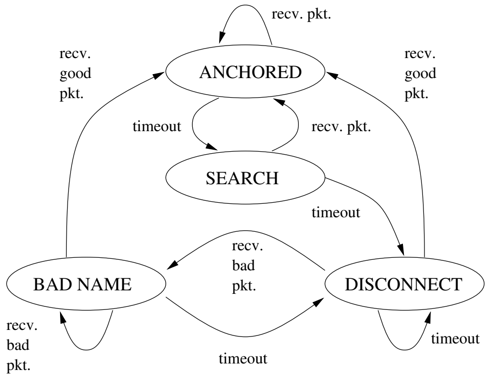

Figure 3: Pulse Operational Modes

4.1 Guards

We use the innate PC card support in FreeBSD to detect the insertion and removal of network interface cards. The **pccardd** daemon uses a database (indexed by card manufacturer and model) to determine which operations to perform on the insertion or removal of a card. We create two utility programs that simply send a message to **pmid** and indicate a change (validation or invalidation) in the *present* characteristic. We configure the database so these programs are executed by **pccardd** on card insertions and removals.

To detect changes in link-connectivity and network addresses **pmid** creates a **pulse** process for every present and enabled LAN interface. Ideally, link-connectivity guards should be placed in the device driver rather than at the network level. However, rather than modify all the drivers we wanted a simpler mechanism for our prototype that would be more driver-independent. Each **pulse** process uses heartbeats (ICMP echo requests and replies) to validate an interface's IP address and link-layer connectivity. Figure 3 describes the various modes **pulse** operates in. In normal connected operation, heartbeats between the mobile host and an adjacent "anchor" host are used to validate connectivity. Once the number of consecutive missed beats passes a threshold (default of 2), **pulse** enters search mode and sends the next heartbeat to the all-hosts multicast group (224.0.0.1) and the first valid reply becomes the new anchor host. If no valid reply is received from the multicast request, **pulse**

```
model cdpd
device 'tun0'
label 'AT&T CDPD'
data
1024 5

model airfone
device 'tun0'
label 'GTE Airfone'
circuit
connection 627
60 328
connection limit 1500
```

Figure 4: Sample Cost Models

enters the disconnect and uses the Berkeley Packet Filter (BPF) [15] to promiscuously listen for all ICMP echo replies. This allows the implementation to distinguish between link-level disconnections (no response at all) and network migrations (responses with incorrect destination link layer addresses). Bad packets refer to packets with incorrect destination link-level addresses. Packets with incorrect sequence numbers, identification fields, or bad checksums are discarded. If any response is detected, the network name associated with the device is invalidated by sending a message to **pmid**. If no response to the ICMP echo request is detected, the link-connectivity characteristic associated with the device is invalidated, again by sending a message to **pmid**. In bad name and disconnected operation, the BPF is used to send out requests since there should be no routes associated with that interface – routes are only attached to available interfaces. If a response is detected, **pmid** is informed so that it may initiate reconfiguration options, e.g., DHCP, to acquire a valid IP address for that interface. FreeBSD's user-level PPP program has the ability to detect the loss of a carrier signal. It then removes all routes bound to the PPP interface. To avoid duplicating effort and increasing overhead, **pulse** processes are only attached to LAN devices.

To monitor costs, **pmid** creates a **costd** process for each interface that has a cost model defined in pmid.conf. Figure 4 shows two sample cost models. The first model represents local (no roaming) cellular digital packet data (CDPD) charges. The keyword "data" indicates that this model is data-based, so fees will be based on data throughput. The line "1024 5" represents a charge of 5 cents per 1024 bytes of data. The second model is taken from a United Airlines' advertisement and represents the cost of using the GTE Airfone [27]. The "connect" keyword indicates a connection-oriented fee, so the cost of this interface depends on the amount of time it is available. The model represents a connection charge of $6.27 ($2.99 + the first minute) with further charges of $3.28 every 60 seconds. The limit indicates an upper bound of $15.00 per call.[6]

The cost model is in an early stage of development and has no support for temporal costs (day vs. evening rates) or handling persistent data (cost built up over multiple connections). The former is necessary for most telephone rates and the latter is necessary to prevent automatic re-dial facilities from creating a large bill by not spending more than a set amount per connection.

User enabling and disabling of devices are detected by adding a guard to the `ioctl` routine inside the kernel to detect SIOCSIFFLAGS commands that affect the IFF_UP interface flag. Changes to this flag notify a PMI support module within the kernel and are propagated eventually to **pmid**.

[6] If you get disconnected, you can redial the same phone number without incurring an additional charge – as long as you contact the operator before the end of the flight to have the additional connections removed from your bill.

Table 3: Guarding Device Availability

Characteristic	Guards
Present	pccardd
Connected	pulse, ppp
NetNamed	pulse, SIOCSIFADDR
Powered	APM (not implemented)
Affordable	costd
Enabled	SIOCSIFFLAGS

The implementation does not currently support any power management guards. Table 3 summarizes the six device characteristics and the modules which guard them.

4.2 Adaptation

When **pmid** receives a device availability event, it initiates adaptation routines starting at the network level. Figure 5 shows the major data structures making up the 4.4BSD network stack implementation. All routes and multicast groups associated with an invalidated interface are removed from kernel data structures by executing a utility program that selectively flushes entries from the routing table. We treat two routes, the default and multicast, with special care. The default route is used when the destination address fails to match any other entry in the route table. FreeBSD supports at most one default route at any time so **pmid** must select one router out of a list of possibilities. The daemon maintains a list of candidates for each interface in order of preference. Ordering follows the preferences used by each discovery mechanism. For example, both ICMP router advertisements and DHCP replies order default route candidates. However, these preferences have no meaning when comparing candidates from more than one interface. In order to evaluate candidates from different interfaces, we have added a policy based on device name and the time at which the device became available. For example, the user may specify that the IBM Ethernet device should be preferred over the dialup PPP interface, and if multiple Ethernet devices are available, the device that became active most recently should be preferred. If an interface used by the default route is invalidated, the route is migrated to the first candidate of the interface with the highest preference. The multicast route (224.0.0.0/8) follows a similar policy, except it gets bound to the interface's IP address. While the addition of a new interface does not cause any routes to be deleted, it may initiate the migration of the default and multicast route to the new interface.

The implementation supports no transport-level adaptation at

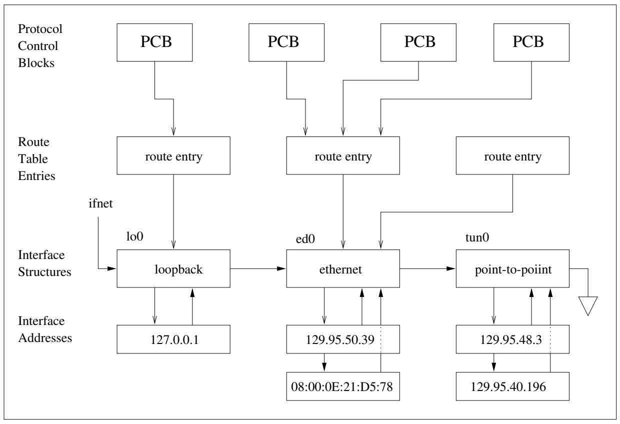

Figure 5: Network Data Structures in 4.4BSD

Table 4: Overhead of the Dynamic Configuration Support

	pmid	costd	pulse
CPU overhead (secs)	2.8u/3.3s	2.8u/2.5s	1.5u/3.2s
CPU overhead (%)	0.17	0.15	0.13
Bandwidth (bps)	N/A	N/A	736

Table 5: Dynamic Configuration Performance

Operation	Time (secs)
add Ethernet	0.47
remove Ethernet	0.55
add WaveLAN	2.96
remove WaveLAN	0.56

this time. We intend to reset local retransmission timers that have begun exponential backoff because of disconnection and use Cáceres and Iftode's "triple ack hack" [3] where multiple acknowledgements are sent to the correspondent host to encourage the remote TCP state machine to enter fast retransmit and recovery mode.

We allow application-level adaptation through the use of a pseudo-device driver. Applications may receive device availability notifications by performing a `select` on the device, which blocks the process until an notification is received. This is an inelegant solution, but allows us to experiment with both synchronous and asynchronous notification mechanisms. Synchronous notifications may be simulated by using `select` before calling `send` or `recv`. Asynchronous notifications may be implemented by forking off a child process that performs a `select` on the pseudo-device and sends a signal to its parent when it wakes up.

5 Evaluation

The evaluation and application experiments were run on a Toshiba T4900CT laptop computer with a 75 MHz Pentium processor and 16 Mbytes of RAM. We wanted to measure both the system overhead and reconfiguration performance. Overhead is measured in terms of CPU and bandwidth usage. To measure overhead we left our daemons running for one hour and recorded how much CPU time (user and system) they consumed over that period. The system was used rather lightly (long telnet-based shell sessions with two local kernel builds) during this hour. We measured **costd** running in data mode, which is usually more CPU intensive since it uses BPF to determine the amount of data passing through the interface, while in connection mode a simple calculation is done every second. We also wanted to know how much bandwidth **pulse** was using. This can be calculated as a 736 bps; 28 byte ICMP message with 4 bytes of data and using a 14 byte Ethernet header twice a second. Table 4 shows the results of our measurements. Note that we measured the overhead of a single process and there may be multiple **pulse** and **costd** processes in a given system. The processor overhead is insignificant, accounting for less than 0.2% of CPU time. **pmid** has the highest overhead, resulting from its periodic (once per second) polling of network interface state maintained in the kernel. The evaluation assumes **pulse** is in a normal anchored state, but there are occasions when multicast ICMP echo requests will be sent out while waiting to acquire a network name. Depending on the number of machines attached to the Ethernet, this can generate a burst of ICMP echo response traffic and result in a high number of collisions.

Table 5 measures the performance of reconfiguring the system when adding or removing LAN cards while connected with a PPP connection. All LAN cards are assigned IP addresses by the configuration files so the system does not have to query a DHCP server or register with a mobile IP agent. The reconfiguration performance of the system is rather poor due to excessive calls to `system` which executes commands in a shell. First, **pccardd** executes `system` to generate a message for **pmid**. After receiving the message, **pmid** executes `system` to run a utility program that removes all route entries attached to a selected interface. When migration is infrequent, these commands result in disk operations.[7] When performed repeatedly, reconfiguration operations complete within 200 milliseconds. The exception is the configuration of the WaveLAN device where the primary culprit is the execution of the `ifconfig` command which is hampered by multiple settling delays used by the driver while configuring the radio modem.

These are the reconfiguration times for our system and do not include the time used by the PC card system (about half a second) or the time it takes to detect a loss in link-connectivity (over four seconds with the parameters we use). It is possible to reduce the detection time by a more aggressive implementation with reduced periods between pulses. This comes at a cost of additional CPU and bandwidth overhead.

6 Application Effects

In Section 4 we described the areas where we did not provide support for adaptation. To determine the support needed by various applications, we use a small suite of programs. Each program makes different assumptions related to device availability. The first application uses long-term TCP connections while the second uses short-term TCP connections. Our third application uses IP multicast to receive real-time video while the final application uses UDP to receive real-time video.

Figure 6 shows the network configuration used by the experiments. The mobile host, MH, can connect via three different physical interfaces: 10 Mbps Ethernet, 2 Mbps AT&T WaveLAN, and 28.8 Kbps POTS PPP. The PPP server lives on network 129.95.40 and is configured to assign IP addresses based on the dialup line. All dialup addresses are allocated from a separate virtual network (129.95.48). The netmask for all subnets in our experiments is 255.255.255.0.

Each experiment starts off with MH connected via both Ethernet and PPP. The Ethernet card is removed for awhile, then the WaveLAN card is inserted. In the beginning, the default route points to RouterB using the Ethernet interace. When the Ethernet card is removed, all route entries associated with that interface are removed. The default route is migrated to the dialup server as it is the only router specified on the only available interface. The multicast route is not migrated, since our configuration file has forbidden a multicast route to ever be bound to tun0, the PPP interface. Note that

[7] Disk power management was not enabled so these operations did not incur the additional several seconds of latency required to spin-up a disk.

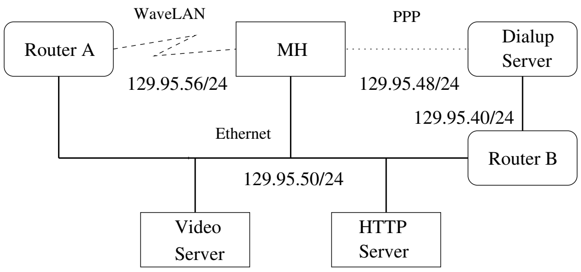

Figure 6: Network for Experiments

multicast groups can still be bound to the interface since the default route uses it and our PPP interface says it supports multicast. When the WaveLAN card is inserted, **pmid** follows the configuration file's policy for configuring 'wlp' devices by first attempting to use DHCP. This succeeds since we have installed the Internet Software Consortium's DHCP server on RouterA. The IP address of the WaveLAN interface, default route, and name server are obtained via DHCP. Since the WaveLAN interface is preferred over the PPP interface, the default route is deleted and a new route is created to use RouterA.

6.1 TCP Sessions

The first application is *telnet*, one of the first TCP/IP programs ever developed. Unfortunately, without network layer adaptation to retain its IP address, *telnet* cannot survive the loss of the interface hosting the local IP address. If we could use the PPP connection as a co-located care-of-address (COA), Mobile IP would enable *telnet* to continue functioning.

6.2 TCP Transactions

Our second application, *Netscape Navigator*, generates HTTP traffic which most often takes the form of a series of short-term connections. Our implementation supports this application quite well, with the exception of indefinitely stalling any data transfers in progress during the removal of the Ethernet interface. Luckily, the stalled transfers can be aborted at the GUI level by clicking the STOP button. New HTTP transactions can be initiated over the PPP interface.

When the WaveLAN card is inserted and the system has reconfigured itself, HTTP transactions in progress continue to use the PPP connection but new connections use the WaveLAN interface. The only problem we encountered was the inability to access the internal web pages using the WaveLAN interface. This is because the Apache web server is protecting those pages using .htaccess files that specify domain names. Because our DHCP server did not provide MH with an IP address with a DNS record, we could not access Web sites. In addition to helping sustain TCP connections across handoffs, Mobile IP retains a host's IP address for authentication purposes. We feel the latter may be more important than the former given the number of utilities that use IP addresses for authentication (hosts.equiv, hosts.lpd, hosts.allow, .rhosts, etc).

6.3 IP Multicast

We selected real-time multimedia applications for our last two applications. Unlike unicast IP, where IP addresses are associated with interfaces, multicast addresses are associated with groups. To receive data sent to an IP multicast group, an interface must join the group using IGMP messages. For mobile hosts, IP multicast provides a level of indirection so the sender never has to know where the receiver(s) are. The network figures this out and routes packets accordingly.

To examine IP multicast, we use *vic*, a video player developed at LBL and U.C. Berkeley [16]. We modified *vic* to rebind the send and receive sockets upon the receipt of an asynchronous mobility notification. This addition of less than 30 lines of code allows *vic* to continue to receive video while moving between Ethernet and WaveLAN interfaces.

While we decided to leave the decision entirely in the hands of the application, Mobile IP does provide two alternatives to joining groups and foreign networks. If the interface has a co-located COA, the interface may join the group by sending an IGMP report to a local router, using the COA as the source IP address. The second alternative uses a bi-directional tunnel back to the home network. That is, the IGMP report is send back to the home network using the home address as the source IP address. None of the choices is optimal for all cases, and the "correct" decision probably requires user input about the characteristics (ttl, bandwidth, security, etc.) of the stream.

6.4 Streaming UDP

For unicast UDP we selected *vcr*, a real-time distributed MPEG video player [4] that we adapted to use our notification mechanism [11]. This video client resides on MH while the video server runs on a workstation attached to the Ethernet network. A reliable TCP connection is used for control messages between the video server and the client controller, while a best effort UDP connection is used to stream video frames and ship feedback messages. The UDP connection has its own feedback policy to implement congestion control mechanisms so it will fairly share the path with TCP connections. The player uses another software feedback policy to adjust the video stream to fit the available bandwidth of the connection. The server adjusts the stream by selectively dropping frames and changing resolutions. For this experiment, the server sends a medium-resolution (256x192x8) video stream at rates up to 30 frames per second. If a display frame rate of 3 frames-per-second (fps) cannot be sustained, the client requests the server to switch to a low-resolution (128x96) stream. If the low-resolution stream surpasses 12 fps, the client asks the server to switch to medium-resolution stream.

On receiving a device availability notification, the player re-establishes the control and data connections between the client and server and resets the feedback system. When reset, the feedback system enters a "exploratory" mode and uses slow-start (exponential growth) policies to quickly determine the available bandwidth of the connection. Figure 7 shows the display frame rate achieved at Ethernet, POTS, and WaveLAN speeds. After the switch to WaveLAN frame rate increases until it passes the 12 fps threshold and switches to the higher resolution stream which can only sustain around 5 fps.

In order to emphasize the usefulness of cross-layer notifications, we compared our strict dynamic reconfiguration with Mobile IP, a system providing IP transparency without notifications. The Mobile IP implementation currently supports handoffs between one class of device, so we simulated heterogenous handoff between foreign agents using a foreign agent (FA) with a WaveLAN interface on one side, and a PPP interface on the other as shown in Figure 8. The home agent was located at Portland State University. The bandwidth between OGI and PSU was large enough (two T-1 lines, approximately 3 Mbps) that the frame rate was still bounded by CPU performance.

Figure 9 shows the display frame rate achieved by the client dur-

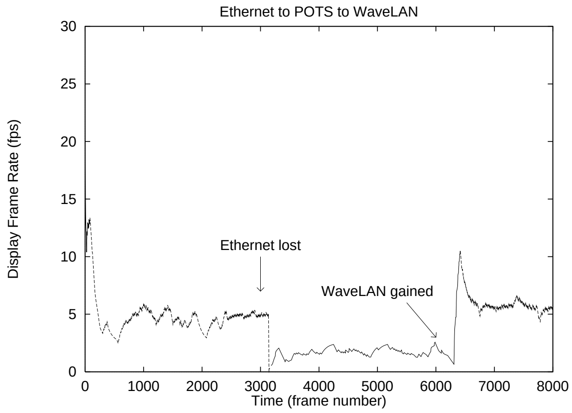

Figure 7: Frame Rate w/ notifications

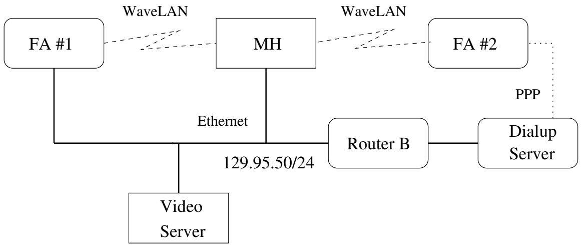

Figure 8: Mobile IP Configuration

ing handoffs to and from the low-bandwidth FA. One goal of the feedback system is to display a smooth picture, so it is very cautious about increasing its bandwidth usage in steady state mode. This is the reason it takes over 2500 frames (83 seconds) after handoff to the HA before switching to the higher resolution image. Compare this to Figure 7, where notifications initiate exponential growth and reaches smooth playback in only 500 frames (17 seconds).

7 Discussion

We have seen how *vcr* exploits higher bandwidth connections significantly better with notifications than without them. The switch to the higher resolution stream occurs within 10 seconds after receiving a notification compared to over 75 seconds without it. We believe that complex applications require cross-layer adaptation. These applications contain many high-level semantics that are hidden from the operating system and network stack. Enabling intelligent adaptation requires these semantics to be transformed into something more meaningful to lower-layers and passed down to them. We use the term *microlanguages* [24] to describe domain-specific languages (DSL) focused on describing application assumptions in a manner meaningful to the application, but also transformable to a DSL more meaningful to a lower layer. For example, *vcr* uses software feedback [4] to monitor the bandwidth between itself and the video server. While the device layer has no understanding of path characteristics, it does understand link-layer characteristics. Allowing the feedback system to register an interest in device availability allows it to guard against drastic changes in path characteristics resulting from network media switching on the local system.

Using notifications in conjunction with Mobile IP would save the application from tearing down and creating another TCP control channel. However, if the network layer adapts first before passing a notification to the application, the video server's data may still

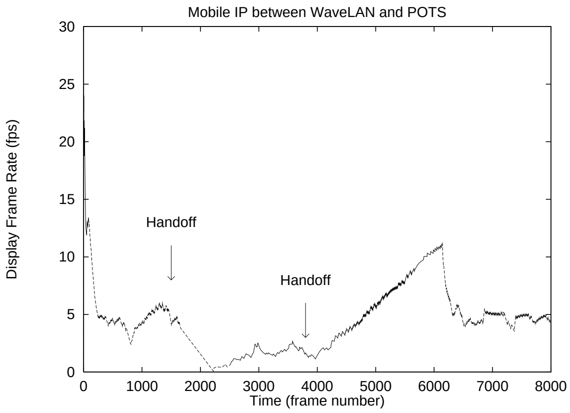

Figure 9: Frame Rate wo/ notification

overflow the low-bandwidth connection before it is informed that the bandwidth to the client has drastically changed. This increases the communication latency because the buffers along the path between server and client have been filled. Physical Media Independence without Mobile IP still benefits a large class of applications using short TCP sessions. These applications are less dependent on the stability of any particular IP address and may have higher level concerns such as available bandwidth, security, or monetary cost.

Another useful aspect of dynamic reconfiguration not demonstrated by our set of applications is the ability to *hibernate* a network-dependent application during periods of disconnection. For example, rather than polling the mail queue every 30 minutes while disconnected, *sendmail* [6] might register interest in being notified when the set of available devices becomes empty. On receiving the notification *sendmail* ceases polling and registers an interest in being notified when the set becomes non-empty. After receiving this notification, *sendmail* checks the queue and delivers any outstanding mail messages. One problem with hibernating processes is their background traffic may interfere with higher-priority foreground traffic. This can be very obtrusive while working over a weakly-connected link. Admission control and network-related priorities might help prevent background traffic from lowering the quality of interactive traffic.

A deficiency in our architecture is that each state machine monitors external events from the environment but ignores state machines associated with other interfaces. There is no global policy for dealing with interactions between interfaces. This prevents the state transitions in one state machine from *motivating* state transitions in others. For example, several PPP implementations support the ability to dial into a PPP server and acquire an IP address on demand, that is, when packets are sent to the interface. So a PPP connection might only be initiated when the last LAN card is removed from a system and an application is attempting to send data without finding an available interface.

8 Related Work

Our project goals are very similar to those of Stanford's MosquitoNet [1] project and Berkeley's Infopad [20] and Daedalus [14] projects. While these groups have focused more on Mobile IP implementations than dynamic reconfiguration policies, they do address many issues that we ignore, such as assumptions about wireless network characteristics and handoff policies. Daedalus has developed a snoop protocol that provides significantly better TCP performance over lossy wireless networks [2]. Stemm's thesis mentions ongoing work toward policies for supporting "vertical handoffs" that determine when to pass an IP address from one type of interface to another [26]. Multiple interfaces are not available at any point in time, just the "best" interface which is selected according to a specified policy.

Dynamic reconfiguration treats Mobile IP as a form of adaptation within the network layer. Fully integrating Mobile IP into our system involves developing additional policies to handle the migration of IP addresses between interfaces. For example, the loss of an interface might migrate the IP address to another available interface, using the available interface's IP address as a co-located care-of-address. This raises some interesting routing policy issues as described by Cheshire [5]. Normal routing policies using metrics related to hop counts may be inappropriate for selecting network routes where path connectivity, cost, and power management are important concerns. One of MosquitoNet's goals is to allow applications to specify their requirements and let the routing policy select the most appropriate delivery mechanism. For example, a *Navigator* session might describe itself as using short-term sessions so (while away from home) MosquitoNet's routing policy might decide to use standard IP with a co-located COA as the IP source address. On the other hand, *telnet* might describe itself as long-term session and by delivered using Mobile IP with bi-directional tunnels. Note that routing decisions are only changed on the *end-system* where the packets are produced and the home agent where packets are re-directed; no modifications to existing routers are required. Determining the appropriate metrics for specifying an applications behavior and how to evaluate them is an area of active research.

The IETF has several working groups working on topics related to dynamic configuration, but they are mostly focused on desktop systems. However we make good use of DHCP, which emerged from an IETF working group, as a mechanism to obtain network information while visiting a foreign network. The Service Location Protocol [28] provides a mechanism for both desktop and mobile hosts to discover resources within their environment including printers, file servers, and proxies.

There is also a significant amount of research adapting applications for mobile operation using existing system mechanisms. Rather than depend on mobility support within the operating system, these applications are designed to work in environments with intermittent connectivity or low bandwidth. POP mailers[8] enable users to retrieve mail during periods of connectivity and read mail after disconnection. Application-level toolkits such as Rover [13] and Wit [29] allow existing applications to be re-engineered for mobility. We believe adaptive applications will perform more efficiently if they cooperate with the operating system. This was reinforced by our experiences developing *vcr*. CMU's Odyssey architecture also supports application-aware applications [25]. Odyssey extends the system call interface, allowing applications to collaborate with the operating system. The new system calls expose resources to application programs by permitting programmers to register a *bound of tolerance* for scarce resources such as network bandwidth, disk space, battery power and cost [21]. When the availability of the resource moves outside the tolerance window, the application is informed via an upcall. Physical Media Independence follows the Odyssey policy of registering interest with resource and providing a notification event. While both architectures support network-aware applications, Physical Media Independence focuses on device availability rather than path-oriented resources such as network bandwidth and latency. Physical Media Independence leaves this to higher layers of software, such as the software feedback system used by *vcr*, and simply provides second-order feedback in the form of device availability notifications.

9 Summary

Physical Media Independence is an architectural framework for dynamically configuring a mobile system in response to changes in the link-layer environment. The framework includes a model for device availability that determines the set of available network devices. Set membership changes are communicated through the system using cross-layer notifications. This allows each module or layer within the system to perform the adaptation best suited to it.

Using a prototype implementation, we ran several varieties of applications to determine their assumptions on the availability of a particular network device. Physical Media Independence without Mobile IP supports short-term TCP connections such as those used by HTTP traffic. The addition of Mobile IP would enable long-term TCP sessions to survive by providing continuous IP address transparency. This allows legacy applications to continue to function across heterogeneous interfaces. The problem is that resource intensive applications make assumptions about an interface's ability in addition to its name. We have shown that these applications adapt in a more efficient manner when the system informs them about changes in the environment.

[8] Mail readers built on top of the Post Office Protocol [19].

Our conclusion is that transparency is an admirable trait, unless it is forced upon applications. In order to properly support mobile computers and mobile-aware applications, a mobile system will have to support multiple levels of adaptation and allow meta-information concerning adaptation policies to propagate between layers. We are continuing to investigate the types of meta-information and programming interfaces necessary to support virtually-connected mobile computing using multiple heterogeneous network devices.

Acknowledgements

We are grateful for the timely and thoughtful comments on this paper from Bikram Bakshi, Dylan McNamee, and Xinhua Zhao. Detailed feedback from our anonymous referees led to improvements in both content and presentation. Shanwei Cen provided us with a mobile-aware version of *vcr*, his streaming media player. This research also benefited from many discussions with members of OGI's Synthetix project, PSU's Secure Mobile Networking project, and Intel's Mobile Technology Lab and Mobile Communications Operation.

This research was supported by the Advanced Research Projects Agency (ARPA) under grant N00014-94-1-0845 and contract MDA904-95-C-5547, Intel Corporation, and Tektronix Inc. Jon Inouye was supported by an Intel Foundation Fellowship. Jim Binkley was supported by ARPA and the U.S. Air Force Rome Laboratory under contract F30602-95-1-0046.

References

[1] BAKER, M. G., ZHAO, X., CHESHIRE, S., AND STONE, J. Supporting Mobility in MosquitoNet. In *Proceedings of the 1996 USENIX Technical Conference* (San Diego, CA, January 1996).

[2] BALAKRISHNAN, H., SESHAN, S., AMIR, E., AND KATZ, R. H. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking* (Berkeley, California, November 1995), pp. 2–11.

[3] CÁCERES, R., AND IFTODE, L. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal of Selected Areas in Communications 13*, 5 (June 1995), 850–857.

[4] CEN, S., PU, C., STAEHLI, R., COWAN, C., AND WALPOLE, J. A Distributed Real-Time MPEG Video Audio Player. In *Proceedings of the Fifth International Workshop on Network and Operating System Support of Digital Audio and Video (NOSSDAV '95)* (Durham, New Hampshire, April 1995), pp. 151–162.

[5] CHESHIRE, S., AND BAKER, M. G. Internet Mobility 4x4. In *Proceedings of the ACM SIGCOMM'96 Conference* (Stanford University, CA, August 1996), pp. 318–329.

[6] COSTALES, B., AND ALLMAN, E. *sendmail*, second ed. O'Reilly and Associates, 1997.

[7] DEERING, S. Host extensions for IP multicasting. Network Working Group Request for Comments: 1112, August 1989.

[8] DEERING, S. ICMP Router Discovery Messages. Network Working Group Request for Comments: 1256, September 1991.

[9] DROMS, R. Dynamic Host Configuration Protocol. Network Working Group Request for Comments: 2131, April 1997.

[10] HALFHILL, T. R. Transforming the PC: Plug and Play. *Byte* (September 1994), 78–94.

[11] INOUYE, J., CEN, S., PU, C., AND WALPOLE, J. System Support for Mobile Multimedia Applications. In *Proceedings of the 7th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 97)* (St. Louis, Missouri, May 1997), pp. 143–154.

[12] INTEL CORPORATION. *Advanced Power Management (APM) BIOS Interface Specification*, 1.2 ed., February 1996. Available at URL http://www.intel.com/IAL/powermgm/.

[13] JOSEPH, A. D., DELESPINASSE, A. F., TAUBER, J. A., GIFFORD, D. K., AND KAASHOEK, M. F. Rover: A Toolkit for Mobile Information Access. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles* (Copper Mountain Resort, Colorado, December 1995), pp. 156–171.

[14] KATZ, R. Adaptation and Mobility in Wireless Information Systems. *IEEE Personal Communications Magazine 1*, 1 (1995), 6–17.

[15] MCCANNE, S., AND JACOBSON, V. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *Proceedings of the 1993 Winter USENIX Conference* (San Diego, CA, January 1993), pp. 259–269.

[16] MCCANNE, S., AND JACOBSON, V. *vic:* A Flexible Framework for Packet Video. In *Proceedings of the Third ACM Conference and Exhibition (Multimedia '96)* (San Francisco, California, November 1995), pp. 511–522.

[17] MCKUSICK, M. K., BOSTIC, K., KARELS, M. J., AND QUARTERMAN, J. S. *The Design and Implementation of the 4.4BSD Operating System*. Addison-Wesley, 1996.

[18] MORI, M. T., AND WELDER, W. D. *The PCMCIA Developer's Guide*, second ed. Sycard Technology, 1995.

[19] MYERS, J., AND ROSE, M. Post Office Protocol – Version 3. Network Working Group Request for Comments: 1725, November 1994.

[20] NARAYANASWAMY, S., et al. Application and Network Support for InfoPad. *IEEE Personal Communications 3*, 2 (April 1997), 4–17.

[21] NOBLE, B. D., SATYANARAYANAN, M., NARAYANAN, D., TILTON, J. E., FLINN, J., AND WALKER, K. R. Agile Application-Aware Adaptation for Mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating System Principles* (Saint Malo, France, October 1997). To appear.

[22] PERKINS, C. IP Mobility Support. Network Working Group Request for Comments: 2002, October 1996.

[23] PORTABLE COMPUTER AND COMMUNICATIONS ASSOCIATION. *Extensions to NDIS 3 for Wireless WANs*, 1995. Available at http://www.airdata.com/pcca/.

[24] PU, C., BLACK, A., COWAN, C., WALPOLE, J., AND CONSEL, C. Microlanguages for Operating System Specialization. In *Proceedings of the SIGPLAN Workshop on Domain Specific Languages* (Paris, France, January 1997).

[25] SATYANARAYANAN, M., NOBLE, B., KUMAR, P., AND PRICE, M. Application-Aware Adaptation for Mobile Computing. In *Proceedings of the 6th ACM SIGOPS European Workshop* (Dagstuhl Castle, Germany, September 1994). Reprinted in Operating Systems Review, Vol. 29, No. 1, January 1995, pp. 52-55.

[26] STEMM, M. Vertical Handoffs in Wireless Overlay Networks. Master's thesis, University of California at Berkeley, 1996. Published as UCB Technical Report CSD-96-903.

[27] UNITED AIRLINES *Hemispheres* MAGAZINE. *GTE Digital Airfone advertisement*, November 1996.

[28] VEIZADES, J., GUTTMAN, E., PERKINS, C., AND KAPLAN, S. Service Location Protocol. Network Working Group Request for Comments: 2165, June 1997.

[29] WATSON, T. Effective Wireless Communication Through Application Partitioning. In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)* (Orcas Island, Washington, May 1995), pp. 24–27.

[30] WICKELGREN, I. J. The facts about FireWire. *IEEE Spectrum 34*, 4 (April 1997), 19–25.

[31] WRIGHT, G. R., AND STEVENS, W. R. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.