January 1997

Physical media independence : system support for dynamically available network interfaces

Jon Inouye

Jonathan Walpole

Jim Binkley

Physical Media Independence: System Support for Dynamically Available Network Interfaces [*]

Jon Inouye
Jonathan Walpole
Oregon Graduate Institute
{jinouye, walpole}@cse.ogi.edu

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

Technical Report CSE-97-001
Department of Computer Science and Engineering
Oregon Graduate Institute of Science and Technology

January 20, 1997

Abstract

Advances in hardware technology has fueled the proliferation of dynamically configurable network interface cards. This empowers mobile laptop users to select the most appropriate interface for their current environment. Unfortunately, the majority of system software remains "customized" for a particular network configuration, and assumes many network characteristics remain invariant over the runtime of the software. *Physical Media Independence (PMI)* is the concept of making assumptions about a particular device explicit, detecting events which invalidate these assumptions, and recovering once events are detected.

This paper presents a model supporting PMI. Based on device availability, the model identifies implicit device-related assumptions made by contemporary network stacks, describes a methodology for making them explicit, and outlines what adaptation should occur when they are invalidated. The model is used to construct a new kernel entity, called the Interface Management Module (IMM), that supports PMI in the FreeBSD operating system. The benefits and limitations of the system are illustrated using a variety of network applications. The results show that transparency is difficult to maintain for all applications because they cache information such as IP addresses and network bandwidth characteristics. We conclude that while low level support for PMI is important, the IMM needs to provide an interface for mapping application-level semantics down to low-level policy decisions.

[*]This research is partially supported by ARPA grant N00014-94-1-0845. Jon Inouye is partially supported by an Intel Foundation Graduate Fellowship. Jim Binkley is supported by ARPA and the U.S. Air Force Rome Laboratory under contract F30602-95-1-0046.

1 Introduction

The goal of Physical Media Independence is desktop equivalence for network management.[1] Laptop computers are inherently more difficult to manage compared to desktop computers because the former are mobile and therefore live in dynamic network environments. Laptop computers are also equipped with hardware technology that allows them to dynamically switch between network interface cards (NIC) without powering off the computer. This flexibility allows laptop users to select the "best" network interface for any particular environment.

Physical media independence is a software response to the challenge offered by technological advances in computer hardware and wireless networks. Compact lightweight notebook computers are widely available, provide competitive price and performance compared to their desktop counterparts, and are easily transported from one location to another. Un-tethered network interfaces such as wireless LANs, packet radio, and cellular modems provide network connectivity without an anchoring wire [19]. Hot swapping technologies, such as PCMCIA [4, 15], allow users to dynamically insert and remove network devices while the computer is operating. These advanced technologies provide the flexibility to dynamically alternate between diverse network devices. However, *software* is responsible for realizing this potential. Software must exhibit various abilities to enable seamless operation while network devices are added and removed. Several examples are listed below:

- *Event detection and identification.* Events affecting the environment must be detected and identified so the system can adapt to them. Detecting some events, such as connectivity, in a distributed environment can be a difficult task.

- *Graceful degradation.* There should be no catastrophic failures! Obviously, there is no way a cellular phone link is equivalent to an Ethernet link, but connectivity should not suffer, only bandwidth.

- *Auto-reconfiguration.* Transparency is important. Adaptation should not require skilled user interaction. By skilled, we mean knowledgeable about system administration.

- *Customizable.* While transparency is often a desirable feature, more sophisticated applications may prefer to direct the reconfiguration. Customizable systems become semi-transparent by allowing application programmers to override the default kernel reconfiguration policies.

These abilities can be illustrated using an example. It is quite easy to imagine the scenario where a student is busy at work at his Ethernet-connected docking station building a new version of xemacs on another machine and using Netscape Navigator to read the New York Times. In the Times entertainment section he notices an advertisement for the new Star Trek movie and visits the Paramount Web site to play the advertised 6 minute MPEG "trailer". He only gets through the first couple of minutes when he notices the time, suspends his laptop, removes it from the docking station, and makes it to the bus stop in time to catch the bus home. During the hour-long bus ride home, he inserts a PC Card [2] packet radio modem and resumes his reading. Reaching his stop, he suspends the laptop once again until after dinner. Replacing the packet radio modem with a cable modem (to improve bandwidth) the student reconnects to the network and resumes watching his movie trailer while investigating several undefined symbols that appeared while linking xemacs.

[1]Desktop equivalence is a somewhat overloaded term. We concentrate on network management. This can be considered the number of calls you need to make to technical support before you can read email, surf the Internet, or transfer files.

[2]A term used to describe a component that adheres to to the PCMCIA standard.

He finishes watching the trailer, adds four more libraries to the Makefile, completes the build, and installs the new version.

This scenario is somewhat possible today, and most of the features should be available within several years.[3] The student has used multiple network devices (Ethernet, packet radio modem, and cable modem) without having to reboot the computer, restart applications, or become super-user and reconfigure by hand.

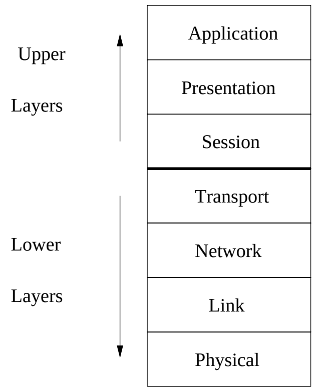

Figure 1: OSI Reference Model

Using the OSI reference model (shown in Figure 1) as a reference, the features in the example above can be supported at different layers. Physical media independence focuses on the link layer, at the interface between the network layer protocols and the network drivers. In the next section we describe related research at other layers of the reference model. Section 3 describes different classes of applications and their requirements for seamless operation. A model supporting physical media independence is presented in Section 4. Section 5 describes an implementation and Section 6 evaluates of the implementation using a diverse suite of application programs. We follow this with a discussion about dynamic reconfiguration in Section 7 and summarize our experiences in Section 8.

2 Related Work

Seamless mobility can be provided by functionality within a module or by a combination from various system components. As a basis for covering related work, we continue to use the OSI reference model. The upper layers usually provide end-user services, like application programs. New applications can be constructed to be *mobile aware* with the assistance of new protocols. POP mailers[4] allow users to retrieve mail during periods of connectivity, and read mail after disconnection. Rather than build new applications from scratch, toolkits such as Rover [12] and Wit [27] allow existing applications to be re-engineered for mobility. The drawback of application level toolkits is they only benefit re-engineered applications.

File systems can support mobility by hoarding files on local disks for use during periods of disconnection. This benefits legacy applications by permitting them to function while disconnected or connected over low-speed links. Coda [24] and OS/2 [26] use a combination of spying and user-level directives to build profiles of which files to hoard. Coda also has a specialized re-integration proto-

[3]The only feature of this scenario unlikely to emerge within the next decade are bus seats where one can type comfortably!

[4]Mail readers built on top the Post Office Protocol [17].

Table 1: Related Work Summary

Level	Systems
Application	Rover, Wit, POP mailers
OS/FS	Coda, OS/2
Network	Mobile IP
Link	Physical Media Independence, Vertical Handoff
Physical	Cellular networks, WaveLAN Roaming

col for weakly-connected operation [16]. However, file systems tend to emphasize data availability rather than network connectivity.

Columbia's Mobile IP [11] investigated the problem of supporting seamless mobility at the network level. Now the responsibility of an IETF working group, Mobile IP [20] allows mobile nodes (both hosts and networks) to retain their IP addresses while traveling. The implications of this are profound. Legacy applications using long-lived TCP connections (such as telnet, rlogin, and remote X11 clients) will continue to retain connectivity. Clients can find mobile information providers no matter where they happen to be located (as long as the client retains connectivity with its home agent). The Daedalus Group at the University of California Berkeley has demonstrated "vertical handoff" which provides mobile IP handoffs across heterogeneous networks [25]. Transfering connections across heterogeneous devices while mobile is also a goal of Stanford's MosquitoNet project [2].

Seamless mobility can also be supported entirely at the physical level. The best example, which already exists, is the cellular telephone network. A point-to-point connection over a cellular modem may be continuously maintained while the mobile computer roams about. Many wireless LANs also support roaming, but only within the same physical network.

2.1 Discussion

Table 1 summarizes various projects using a layered model quite similar to the OSI model. The examples show that seamless mobility can be supported at different system layers but often require communication between the layers for best performance. Application level toolkits that make use of existing network implementations assume they will provide the best connectivity possible at any given moment in time. This is not always the case, and we discuss such situations in Section 4.

Rather than simply benefiting re-engineered applications, hoarding file systems benefit legacy applications. However, due to their coarse granularity (files) and lack of application level semantics, they can be less efficient than re-engineered applications that have better knowledge of the granularity, integrity, consistency, and priority of the working data set.

Mobile IP and cellular networks provide seamless network and link layer connectivity respectively, but being low-level mechanisms, they do not know about higher-layer semantics or assumptions. There are known performance problems associated with certain transport protocols running over mobile IP and wireless networks. Cáceres [5] demonstrated how handoffs and lossy wireless networks can affect reliable flow-controlled transport protocols like TCP. Developed primary on fixed and wired hosts, TCP's congestion control policies politely reduce throughput when packets are not acknowledged within a given time frame. This can inappropriately limit throughput after cell handoffs or while operating on lossy wireless links. To alleviate these problems, various strategies have been used. Loss of throughput during cell handoff has been reduced by taking advantage of features in modern TCP implementations to restart TCP's retransmission timer [5] and multicasting packets to adjacent cells [3]. Bandwidth over lossy wireless networks has been improved by the use of a snoop protocol [3] where a local agent retransmits packets over the wireless connection.

File systems can also make assumptions about the characteristics of the underlying network. For example, you can move a NFS client away from its home network. Using Mobile IP, you can connect it to a foreign network and tunnel packets to and from the NFS server. Or you can dialup a modem on the server using a cellular modem and establish a point-to-point connection. While these solution provide connectivity, most applications making heavy use of the file system will notice a significant difference between local and remote operation!

We do not want to claim that any particular level best supports mobility since each has its own advantages and disadvantages. Rather, we view the solution of seamless mobility to include support at, and cooperation between, *all* layers. In this paper we present the role we see for physical media independence and how it might be supported in contemporary systems.

3 Requirements for Physical Media Independence

A system supports physical media independence if it adapts gracefully to changes in the available physical media, such as the addition and removal of network interfaces. Conceptually, physical media independence presents a *virtual link-layer interface* to higher layer protocols. This virtual interface is available as long as some physical interfaces are available. It becomes unavailable only when no physical interfaces remain available.

The definition of PMI presented above leaves room for a range of possible PMI solutions, each preserving a different degree of transparency for higher-level connections. One criterion for distinguishing among PMI solutions is their ability to establish new connections following changes in physical interfaces. Another is their ability to migrate existing connections transparently among physical interfaces. These characteristics dictate the class of higher-level applications that are well-supported by a particular PMI solution.

- *Fine-grain information clients.* An information client obtains information from remote information sources. A client is fine-grain if its reliance on connection-oriented protocols is "short". World Wide Web browsers are a good example of fine-grain clients. While HTTP transactions are based on connection-oriented transfers, starting a new transfer does not require the use of the same link, network, or transport address on the client-side. An aborted transaction is not very obtrusive unless it is long lived, such as obtaining a 20 MB MPEG video over a 14.4 Kbps modem. To support fine-grain information clients, a physical media independent system only needs to be able to support new connections over a new interface. Connections are terminated when interfaces are removed or the network address changes.

- *Fine-grain server.* An information provider requires that it clients be able to locate it no matter where is happens to be attached to the network. A more responsive DNS system might be able to support fine-grain servers without supporting seamless connectivity. Details of such as system are beyond the scope of this paper. Connections are terminated when interfaces are removed or the network address changes.

- *Continuous connectivity* Connections bound to an interface are migrated to other available interfaces, or suspended and resumed when an interface becomes available. Applications that require this include long-lived, session-based, interactive applications such as remote login shells and telnet-based on-line library database searches.

Table 2: Device Characteristics

Present	Device is physically attached
Power	Power is available
Connect	Link-level connectivity
Bound	Network name is bound to device
Price	Cost of using device is within budget
Enabled	User allows the device to be enabled

It is our opinion that mobile hosts will rarely be primary information providers. When a mobile host is disconnected or suspended, it is not able to service any requests for information making it ill-suited for such applications.

4 A Model to Support Physical Media Independence

A network interface is the software abstraction of a physical network device. Interfaces can be *available* or *unavailable*. 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 the physical network device. Packets received by the physical network device are passed by the device driver up to the network layer. Packets do not flow across an interface in either direction when it is unavailable.

We define six characteristics that determine when an interface is available.[5] An interface is *present* if both the hardware device and the software driver exists. An interface is *connected* if there exists a connection at the link layer.[6] We only consider link layer connectivity here, not network connectivity. So a twisted pair Ethernet interface card connected to an isolated hub would be considered connected. An interface is *bound* when there exists a binding to a higher level name, such as a network address. An interface is *powered* if the network device has enough power to function. Because many wireless devices may be expensive to use, we also associated a *price* with each interface. An interface is *enabled* unless it is specifically disabled by user-level directives such as `ifconfig`. Equation 1 states the an interface I is available if the conjunction of its six characteristics are true.

$$Available(I) \vdash I_{present} \wedge I_{power} \wedge I_{connect} \wedge I_{bound} \wedge I_{price} \wedge I_{enabled} \tag{1}$$

The truth values of characteristics may be changed by a variety of events. We denote E_c as an event changing the value of characteristic c. Hardware interrupts are generated when PCMCIA devices (PC cards) are inserted or removed. Theses interrupts are serviced within Card Services[7] where they are passed on to the PMI Manager via $E_{present}$ messages. Many device have the ability to detect changes in link layer connectivity and pass this information on to their drivers. For example, serial devices detect the loss of a data terminal ready (DTR) signal, twisted pair Ethernet devices detect loss in link integrity, and WaveLAN devices monitor signal strength. Drivers which either actively probe or handle device interrupts generate $E_{connected}$ when link layer status changes are detected. Changes in binding and device status are initiated through the use of system calls

[5]These six are relevant for our domain, and we concede the likely existence of other useful characteristics, such as security, for other domains.

[6]This characteristic is required since many users disconnect cables rather than remove cards. The latter operation is often unnecessarily difficult to do on some laptops.

[7]This software module supports PCMCIA cards.

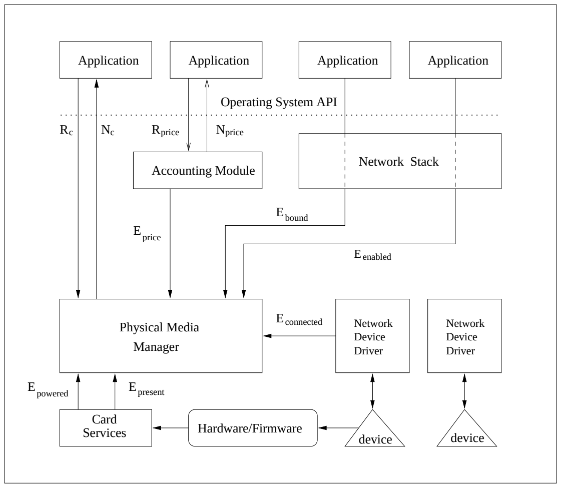

Figure 2: Characteristic Modifying Events

using the operating system application programming interface (API). These system calls generate E_{bound} and $E_{enabled}$ messages. We follow the PC Card '95 Standard where Card Services (rather than the device drivers themselves), generate E_{power} messages in response to power management operations. Both applications and other system modules can issue requests for notification, R_c, if characteristic c changes. This includes device availability. The PMI Manager will respond with a signal notification, N_c on receipt of an E_c message. Figure 2 illustrates the model in a typical environment. Note that PCMCIA insertion and removal events will be masked by power suspension, and we assume the Card Services will generate the appropriate messages when power is restored.

Contemporary network stack implementations often make assumptions about the availability of an interface and its characteristics. While these assumptions are correct for traditional fixed systems, they are invalidated by emerging mobile systems. Many of these assumptions take the form of binding associations where an interface is bound to another structure. Before a network layer protocol can use an IEEE 802 interface, it must establish a binding between the network and link layer address. For performance reasons, these bindings are often cached. Some protocols assume that bindings rarely change once they are established which obviates the need to maintain strong consistency between binding caches. ARP [21] maintains consistency using timers; bindings are cached until they expire. There is no mechanism to invalidate an ARP binding though such mechanisms have been proposed [13]. Routing daemons bind routes to interfaces. Routes are often cached in protocol control blocks to avoid searching the routing table for each packet. Connections are another type of binding association. Transport protocols like TCP [22] represent a connection as a binding between two endpoints. These endpoints are usually network addresses, which are in turn bound to interfaces. Other possible bindings include quality of service (QoS) agreements, scope, group management (IGMP), and security.

While these assumptions are not always appropriate, we do not want to insert interpretation code to validate assumptions before using the code specialized on those assumptions. One worst case scenario would be to probe the device to determine whether it is present, powered, and connected each time its interface is referenced. This complicates the code and degrades performance. Our solution is to continue using assumptions, but make them explicit. This requires placing *guards* in locations that detect events invalidating assumptions. Because these events are relatively infrequent compared to the number of times the assumptions are used, making assumptions explicit has a negligible impact on performance.

When guards are executed, they trigger *replugging* actions which must invalidate the assumptions protected by the guard. This involves re-binding the data structures that assume a static network interface. Bindings associated with that interface can be reconfigured to adopt another available interface. For example, routes can be redirected to a different interface. If no other interfaces are available, then new requests for external connection are rejected and existing connections are suspended. If an entity has requested to be notified of changes in device unavailability, then it is sent a notification event.

We depend on Mobile IP [20] to hide changes in network bindings from higher level protocols. Mobile IP supports this using a layer of indirection provided by intermediate routers (called the Home Agent and the Foreign Agent) that forward packets to and from the mobile host.

5 Implementation

We implemented a subset of the model in the FreeBSD operating system.[8] We chose FreeBSD due to the freely distributed nature of the code, low cost, commercial quality of the network stack, documentation on the network stack, and it booted easily on our platform, a Toshiba T4900CT laptop computer. The disadvantages of choosing FreeBSD are the alpha nature of the PC card and APM [10] support. The FreeBSD network code is derived from the 4.4BSD-LITE distribution [28].

Table 3: Guarding Device Availability

Characteristic	Guards
Present	PCMCIA callbacks (intercepted in CS)
Bound	ioctl (SIOCSIFADDR command)
Connected	network monitor events
Powered	APM events (intercepted in CS)
Price	cost monitor events
Enabled	ioctl (SIOCSIFFLAGS command)

Table 3 list the six characteristics of an available device and the events which change their state. When cards are inserted or removed, PCMCIA socket controllers generate interrupts and the Card Services (CS) module probes to controller's registers to determine what event caused the interrupt. Changes in the bound and enabled state are detected by guards in the `ioctl` calls that set an interface's flags and addresses. Changes in link layer connectivity are much harder to detect since there is no uniform standard for this ability. The IEEE 802.3 10BASE-T standard defines a twisted-pair Ethernet link integrity test but does not require all such devices to implement it. Many wireless LAN cards provide mechanisms for detecting the signal strength of other cards operating within its range. However, these features may not be available unless the device driver exports the capability. In cases where drivers lack the ability to detect changes in link layer connectivity, we use a network monitor to detects connectivity changes at the network layer. The network monitor lives outside the kernel in a user-level process. Periodically, it uses `ioctl` calls to build an image of the available interfaces. For each interface that does not support a link layer connectivity check, a connectivity node (CN) is selected using the following criteria. For a point-to-point interface, the node at the other end of the connection is chosen. For an 802-type interface, the network monitor

[8]We used version 2.1-RELEASE with the pccard-test-960414 patchfile from the FreeBSD Nomad group.

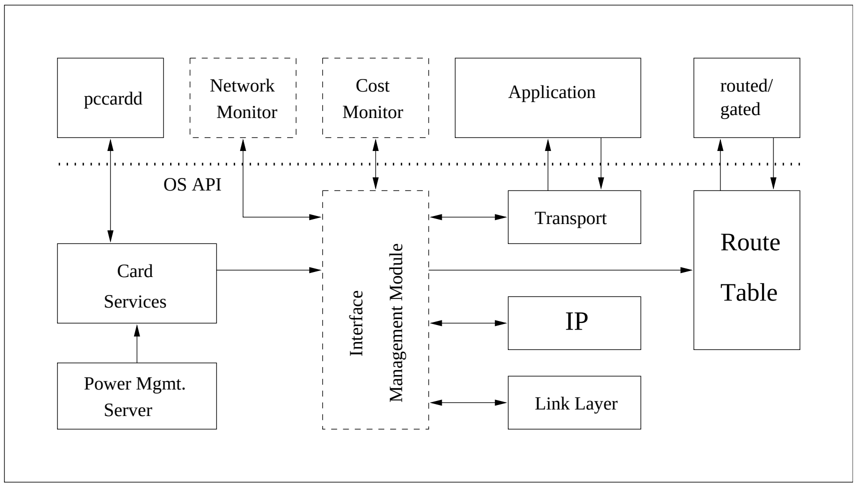

Figure 3: Physical Media Independence in BSD Unix

uses the network mask and IP address of the interface to create a subset of the IP address space to search for a CN. At a configurable periodic interval, ICMP request messages are sent to each CN. If N pings in a row are lost, the monitor uses an `ioctl` call to signal that connectivity has been lost.

This design suffers from having to retain a host route to the CN, which also requires the interface to remain "available" for the network monitor to continue pinging it. Without this, it is impossible to discover re-establishment of link-level connectivity. This design also does not work for a mobile host equipped with mobile IP. On a foreign network, a mobile host continues to use its own IP address.[9]

We are currently re-implementing the network monitor to use the BSD packet filter to send out and receive ICMP packets. A better solution would be to standardize such a capability in the device driver interface and network devices.

The cost monitor design is still in a state of fluctuation and has not been used in any of the experiments. The basic idea is that the user specifies the "cost" associated with each interface. The cost monitor uses this specification to keep track of how much network interfaces are costing. For "connection-oriented" interfaces like cellular phones, cost is monitored by the amount of time the interface is available. "Data-oriented" interfaces like CDPD modems are monitored using the BSD packet filter [14] which monitors the data flow through the interface in determining the cost.

Figure 3 shows the block diagram of the implementation. The Interface Management Module (IMM) lives inside the kernel.[10] The IMM receives events about changes in an interfaces traits. When an interface becomes available or unavailable, a replugging action is performed to reconfigure the network stack for the new environment.

5.1 Replugging

The 4.4BSD-LITE implementation represents interfaces using an `ifnet` structure. When an interface becomes unavailable all bindings, i.e., references, to its `ifnet` structure must be updated. Figure 4 illustrate some of the primary data structures used in the 4.4BSD-LITE implementation.

[9]Mobile IP also allows the host to acquire a care-of-address (COA).

[10]Actually, there is also a significant portion that lives outside the kernel.

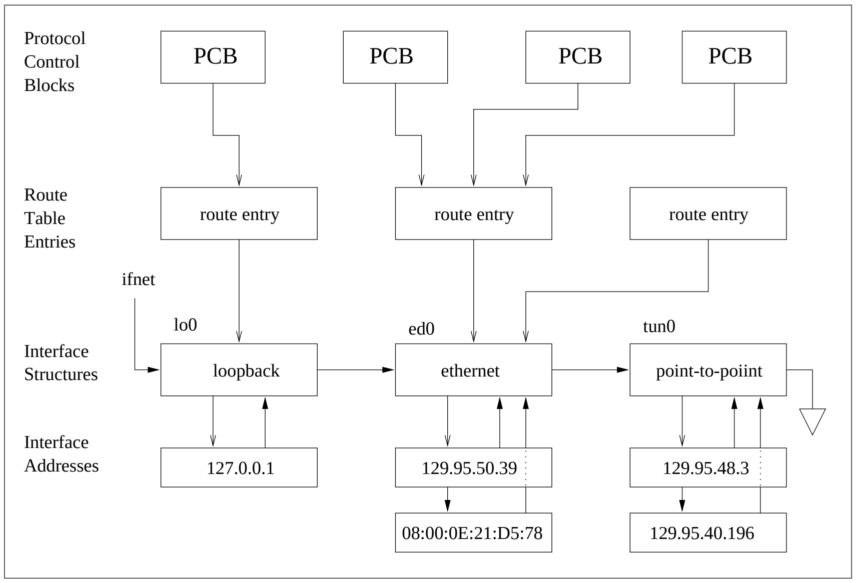

Figure 4: Network Data Structures in 4.4BSD-LITE

We used `grep` on the FreeBSD sources to track down the various possible bindings to an `ifnet` structure. We found direct references from various other kernel data structures including interface address (`ifaddr`) back pointers, device-specific interfaces, the Berkeley packet filter, route table entries, multicast address structures (for IGMP), and virtual interfaces (for multicast routing). There are also many indirect references through these references. Pattern matching tools like `grep` are a poor match for this type of task and we are continuing to find new bindings that need to be replugged. We have collaborators developing both compiler-based and run-time tools to automatically track references (both direct and indirect) to particular data structures [6].

Interface flags. The `ifnet` structure is extended with additional flags representing the six device characteristics. The IMM toggles these characteristics on reception of E_c messages. After updating the flags, a check is performed to determine whether the availability status should be changed. When an interface becomes available we do not need to perform any additional operations beyond those performed by FreeBSD.[11] When an interface becomes unavailable, its `ifnet` structure is marked down by clearing the IFF_UP flag. Operations to set and unset the IFF_UP flag are modified to use the IFF_ENABLED flag instead.

Route table entries. All routing table entries associated with that interface are invalidated. This includes ARP entries for neighbors, static entries (like the default route), and entries with gateways using that interface. Static entries are preserved over the life of the operating system. For example, if a static route is removed because an interface becomes unavailable, it is restored when the interface becomes available and has the same name binding. Messages are sent to the routing socket to notify the routing daemon that an interface has been marked down.

[11]It may be useful for an existing connection to migrate to new interface for better bandwidth, QoS guarantees, etc.

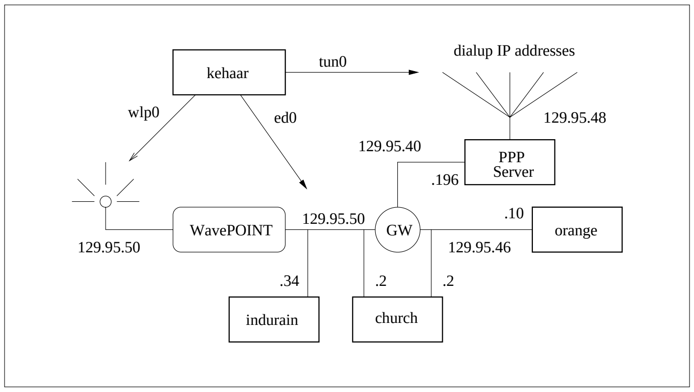

Figure 5: Network for Experiments

IP multicast groups. Each interface may be associated with IP multicast groups as specified by RFC 1112 [7]. We needed to decide whether or not to migrate group membership information to another available interface. Migration is not always possible since not all interfaces support multicasting and there are other complications. Our first implementation migrates group information by searching the list of interfaces until one supporting multicast is found. This does not always lead to a better solution as our experiments illustrate, and we are reconsidering this decision.

6 Evaluation

Figure 5 shows the network configuration used by the experiments. There are five hosts, represented by rectangles, on three physical networks connected by a Cisco router (GW). An AT&T WavePOINT bridge [1] provides subnet 130.95.50 with WaveLAN access [23]. The mobile host, `kehaar`, is running FreeBSD while `indurain`, `church`, and `orange` are running HP-UX 9.03, Solaris 2.3, and SunOS 4.1.3_U1 respectively. The PPP server lives on network 130.95.40 and is configured to assign IP addresses based on the dialup line. All dialup addresses are allocated from a separate virtual network (130.95.48). The netmask for all subnets in this example is 255.255.255.0.

Several applications are used to evaluate the system. `kehaar` is running **Mosaic** (World Wide Web browser accessing pages on `church`), **httpd** (an Apache HTTP server), **nv** (MBONE network video conference tool using IP multicast), **ttcp** (network benchmark program that sends UDP packets to `orange`), and **rlogin** (login session to `indurain`). `church` is running another HTTP server, `orange` is running **nv** and **ttcp**, and `indurain` is running Mosaic (accessing Web pages on `kehaar`). `kehaar` can use three network interfaces: a serial device for PPP connections (tun0), Ethernet PC Card (ed0), or WaveLAN PC Card (wlp0). The Ethernet and WaveLAN devices are hot swappable.

6.1 Experiments

We evaluate our initial implementation of a physical media independent system using a series of simple experiments. In the first experiment, `kehaar` starts with an Ethernet connection. The Ethernet card is removed and a connection to the PPP server is established using the serial interface and a 14.4 Kbps modem. Dynamic routing is being used in passive mode (`routed` with -q option) and the initial default route is set to the Cisco router (GW). The PPP client will set the default route to point to the other end of the point-to-point interface if one does not exist.

6.2 Results

Table 4 summarizes the behavior of the three applications when switching from the Ethernet to the PPP interface. Since our mobile IP implementation is not finished, requests to **httpd** will never reach it when `kehaar` acquires a new IP address. If `kehaar` had registered a dialup IP addresses to its host name (multi-homed node) then an intelligent client that steps through each IP address in a `hostent` structure will eventually connect to `kehaar`, if `kehaar` is connected. Due to the hard assignment of IP address to dialin lines on the PPP server, that is not an option in this case and the approach is not scalable without a change to the Domain Name Service (DNS) system.

The **rlogin** application faces a similar fate. While the IMM allows packets to flow out the interface, the source address of the packets remains 130.95.50.39. The return packets from `orange` never reach `kehaar` and the session becomes unresponsive. While it is possible to alter the packet's source address on `kehaar`, the other endpoint would reject them since it has already established a binding to address 130.95.50.39 and there is no mechanism to re-establish a new binding except for aborting the connection and creating a new one. Mobile IP would allow **rlogin** to continue operating.

The new IP address does not produce a crippling affect on information clients. If the **Mosaic** browser was performing a transaction when the interface was removed, the connection will stall but this can be aborted at the GUI level by clicking on the spinning globe. New HTTP transactions can be initiated over the PPP interface. In the original FreeBSD system, an ARP entry to `church` is cached in the routing table so all new TCP connections to that host will fail with the message "host is unreachable" until the entry times out.[12] Since the IMM scrubs the ARP entries associated with neighbors of an unavailable interface, this is not a problem in our system. In this case, Mobile IP may provide less than optimal performance since the data sent in response to the request would be forwarded from the home agent. No such indirection exists if the `kehaar` uses the new IP address. There is current research in avoiding Mobile IP's "triangle routing" but it still requires network connectivity to the home agent for authentication before packets are forwarded [18].

The `ttcp` application fails in both cases, but due to different reasons. When a PC Card is removed, FreeBSD marks the interface down but doesn't touch the routes. UDP packets sent to `orange` will find a route but notice that the interface used by the route is down so a "Network is down" error will be generated. The IMM removes routes, so while no other external interfaces are available, UDP packets will not even find a route to `orange` so a "No route to host" error will be generated. While UDP is connectionless, applications that assume constant connectivity will not survive periods of disconnection while interfaces are swapped. If the user had started up the PPP server *before* removing the Ethernet card, an interface would always have been available, even though some packets might have been lost.[13]

nv is a video conferencing tool, but `kehaar` is sending output from the X11 display since it has no video capture hardware. Both IP multicast send and receive ability is lost during migration to the PPP interface. In the case of unmodified FreeBSD, the Ethernet interface is marked down and all packets sent to that interface will be discarded. With the IMM, the IP multicast group is migrated from the Ethernet device to the PPP device (whose flags indicate it has multicast capability). Unfortunately, the PPP server does not support IP multicast ability and appears to discard packets when it receives them. In both cases, packets are discarded but with the IMM migrating the group packets are sent out the PPP connection. While we only using a 64 Kbps

[12] The default expiration time of an ARP entry is 20 minutes.

[13] In the current implementation, mbufs attached to a device's output queue are not migrated to another available device. It is possible to migrate these buffers, but the link layer headers in them require transformation when migrating to another type of network interface.

stream this is enough to overwhelm a 14.4 Kbps connection and cause the UDP buffer to overflow. This results in less bandwidth available for other connections, and a new rlogin connection will notice this! Transparently migrating the multicast group results in no additional functionality and worse overall performance. We are currently debating whether or not IP multicast groups should be migrated, and if they should migrate, under what conditions is this acceptable.

Table 4: Experiment 2 Results

Application	FreeBSD 2.1-RELEASE	FreeBSD with IMM
Mosaic	HTTP requests to `church` block	normal operation
HTTP server	becomes unavailable	becomes unavailable
ttcp	exits when sendto returns error ENETDOWN	exits when sendto returns error EHOSTUNREACH
nv	fails but no packets sent over PPP connection	fails but packets sent over PPP connection
rlogin	connection unresponsive	connection unresponsive

7 Discussion

We have presented a model supporting physical media independence and an initial implementation of the model. The model is based on the availability of interfaces and follows an event-driven approach. An interface is represented as a state machine where a state is defined by its characteristics (present, connected, etc). A deficiency in our model is that it only detects events that cause state transitions. What our models fails to accomplish is *motivate* state transitions. While the user is the only one capable in creating $E_{present}$ events, there are sufficient mechanisms that allow the system to manufacture the other characteristics. Moving the responsibility of initiating these events away from the end user is closer to the desirable trait of auto-configuration.

7.1 Dynamic Reconfiguration

We use the term *dynamic reconfiguration* to imply the system has the responsibility of motivating and manufacturing state changes in device availability. 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. The device driver and PPP client (which contains the dial in and authentication information) are tightly integrated to provide the functionality of a virtual connection. Combining this capability with a cellular modem and a intelligent power managed device provides the system with the mechanisms to change the device's characteristics without user intervention.

We break down this functionality into two parts. The first task is to determine *when* a device should attempt to change its availability status. This involves knowing a device's capabilities and being able to intelligently select which devices should be made available and which should be made unavailable. It also involves the determining the conditions that should initiate an interface's migration to a new state. For example, if you are flying cross-country, you can connect to home PPP server via a modem and a GTE AirFone. However, this connection will be rather expensive and therefore should not be transparent to the user footing the bill!

The second task is to determine *how* to initiate a change in a device characteristic. There are many ways of acquiring a binding to an IP address. Mobile IP allows you to use the same IP address of the host, no matter where it is connected. Router advertisement packets inform an unbound interface about the current network address and available routers on that network [8]. Interfaces can then automatically acquire a network address using either the MAC address[14] or the Dynamic Host Configuration Protocol (DHCP) [9].

One active area of research we are investigating is a high-level specification language that allows users to request certain guarantees from the system and allows to system flexibility in meeting these goals. This may help pass user-level information down to the IMM to help in policy decisions such as choosing a default route, resolving power vs. connectivity conflicts, and selecting an appropriate monetary budget for network connectivity.

8 Summary

Physical media independence (PMI) hides significant link layer events from higher layer protocols unless they specify an interest in them. We have defined the concept of physical media independence and its role in supporting, at the link layer, the more ambitious goal of seamless mobility. An event-driven model based on device availability is presented to support physical media independence in contemporary systems that retain assumptions about the persistence of network interfaces. This model makes such assumptions explicit by placing guards at locations where these assumptions may be invalidated. When an event invalidating an assumption is detected, a replugging operation is performed to invalidate or correct the bindings (assumptions) in use.

We described a partial implementation of the model using the 4.4BSD-LITE network stack in the FreeBSD operating system. A simple experiment using a variety of applications has shown the advantages of the PMI-enhanced FreeBSD has over the original. However, there is one case where performance suffers because the link layer does not understand higher level semantics such as necessary bandwidth. The initial implementation is designed to react to events, but an example was given where an implementation may want to manufacture events and this is an area of ongoing research. Physical media independence will not, by itself, provide seamless mobility. But it is a useful and necessary tool in achieving it.

References

[1] AT&T. *WavePOINT Installation and Operation Guide*, 1995. ST-2124-04.

[2] 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).

[3] 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.

[4] Bryan, J. PCMCIA: Past, Present, and Promise. *Byte* (November 1994), 65–72.

[14] Both IPv6 and OSI allow a MAC address to be converted into a network address.

[5] 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.

[6] Consel, C., and Noel, F. A general Approach for Run-Time Specialization and its Application to C. In *Processings of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'96)* (St. Petersburg Beach, Florida, January 1996).

[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: 1541, October 1993.

[10] Intel Corporation. *Advanced Power Management (APM) BIOS Interface Specification*, 1.2 ed., February 1996. Use URL http://www.intel.com/IAL/powermgm to obtain a copy.

[11] Ioannidis, J., and Maguire, Jr, G. Q. The Design and Implementation of a Mobile Internetworking Architecture. In *Proceedings of the USENIX 1993 Winter Conference* (San Diego, CA, January 1993), pp. 491–502.

[12] 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.

[13] Malkin, G. ARP Extension – UNARP. Network Working Group Request for Comments: 1868, November 1995.

[14] 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.

[15] Mori, M. T., and Welder, W. D. *The PCMCIA Developer's Guide*, second ed. Sycard Technology, 1995.

[16] Mummert, L. B., Ebling, M. R., and Satyanarayanan, M. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles* (Copper Mountain Resort, Colorado, December 1995), pp. 143–155.

[17] Myers, J., and Rose, M. Post Office Protocol – Version 3. Network Working Group Request for Comments: 1725, November 1994.

[18] Myles, A., Johnson, D. B., and Perkins, C. A Mobile Host Protocol Supporting Route Optimization and Authentication. *IEEE Journal on Selected Areas in Communication 13*, 5 (June 1995), 839–849.

[19] Padgett, J. E., Gunther, C. G., and Hatori, T. Overview of Wireless Personal Communications. *IEEE Communications Magazine 33*, 1 (January 1995), 28–41.

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

[21] Plummer, D. Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. Network Working Group Request for Comments: 826, November 1982.

[22] Postel, J. Transmission Control Protocol. Network Working Group Request for Comments: 793, September 1981.

[23] Rash, Jr, W. WaveLAN: A Network with No Strings Attached. *Byte 16*, 6 (June 1991), 294–296.

[24] Satyanarayanan, M., Kistler, J. J., Mummert, L. B., Ebling, M. R., Kumar, P., and Lu, Q. Experiences with Disconnected Operation in a Mobile Computing Environment. In *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium* (Cambridge, Massachusetts, August 1993), pp. 11–28.

[25] 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.

[26] Tait, C., Lei, H., Acharya, S., and Chang, H. Intelligent File Hoarding for Mobile Computers. In *Proceedings of the First Annual International Conference on Mobile Computing and Networking* (Berkeley, California, November 1995), pp. 119–125.

[27] 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.

[28] Wright, G. R., and Stevens, W. R. *TCP/IP Illustrated, Volume 2: The Implementation.* Addison-Wesley, 1995.