



MOSS: A Mobile Operating System Substrate

Citation

Chen, J. Bradley, H.T. Kung, and Margo Seltzer. 1995. MOSS: A Mobile Operating System Substrate. Harvard Computer Science Group Technical Report TR-14-95.

Permanent link

<http://nrs.harvard.edu/urn-3:HUL.InstRepos:25620449>

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA>

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. [Submit a story](#).

[Accessibility](#)

MOSS: A Mobile Operating System Substrate

J. Bradley Chen, H. T. Kung, and Margo Seltzer

TR-14-95



Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

MOSS: A Mobile Operating Systems Substrate

A White Paper

J. Bradley Chen, H.T. Kung, and Margo Seltzer

Harvard University

Division of Applied Sciences

Cambridge, MA 02138

May 1995

Abstract

The Mobile Operating System Substrate (MOSS) is a new system architecture for wireless mobile computing being developed at Harvard. MOSS provides highly efficient, robust and flexible virtual device access over wireless media. MOSS services provide mobile access to such resources as disks, CD ROM drives, displays, wired network interfaces, and audio and video devices. MOSS services are composed of virtual circuits and virtual devices. Virtual circuits (VCs) on wireless media support the spectrum of quality-of-service (QoS) levels required to cover a broad range of application requirements. Virtual devices implement resource access using VCs as their communication substrate. The tight coupling of network code and device implementations makes it possible to apply device-specific semantics to communications resource management problems. MOSS will enable mobile software systems to adapt dynamically to the rapidly changing computing and communications environment created by mobility.

1. Introduction

Mobile, wireless computing systems, like systems based on traditional wired networks, should provide access to remote (global) resources. There is nothing fundamentally new about remote resource access; it has been a part of computing since the introduction of wired computer networks. The key factor that differentiates mobile, wireless computing from current models of networked computing is that in the mobile, wireless case the environment is constantly changing. More specifically, the quality of the network connection varies significantly and can change frequently. Also, the resources available to a mobile computer in its local environment are constantly changing. The challenge in mobile computing is to respond gracefully to the changes in the computing environment created by mobility. This challenge motivates our development of MOSS.

MOSS is a new system architecture for networked, mobile computing being developed at Harvard. The MOSS architecture is motivated by five factors that differentiate computing in a mobile, wireless environment from computing on stationary, wired hosts:

- Mobile hosts are resource constrained.
- Resources in the local, wireless environment are constantly changing.
- Bandwidth is constantly changing and sometimes inadequate.
- Link errors are significant and change over time.
- Changes in connectivity are a normal occurrence.

This paper is available from the Center for Research in Computing Technology, Division of Applied Sciences, Harvard University as technical report 95-14-95.

The goal of MOSS is to create a software infrastructure that provides access to remote resources while adapting to the changes in connectivity, bandwidth, latency, and error rates occurring in the networks used by mobile computing systems.

The current generation of communications protocols and operating systems is based on two decades of experience with stationary machines and static network configurations. For example, in today's systems, packet loss is always attributed to congestion, and network topology is static and inflexible. Neither of these conditions holds in a mobile environment. Although many current research projects are exploring software structure for mobile computing, they invariably start with legacy protocols and systems, then attempt to add support for mobility by interposing additional software layers to overcome the inappropriate assumptions of legacy systems. We believe that this approach is flawed and will lead to complex and inefficient systems. In MOSS, legacy system components are replaced with new abstractions that provide a sound foundation for mobile applications. The MOSS system design is created precisely to meet the needs of mobility.

The remainder of this document is organized as follows: First, we give an overview of the MOSS architecture. In the next two sections we focus on technical issues relating to virtual circuits and virtual devices, followed by an overview of the MOSS services we plan to implement. We close with a review of relevant related work and some brief conclusions.

2. MOSS Architecture

The architecture of MOSS is based on two key abstractions: virtual circuits and virtual devices. Virtual circuits provide a low-level mechanism for managing communications resources. The use of virtual circuits is motivated by new network resource management problems resulting from mobility and mobile applications. The key property of virtual circuits for MOSS is that they are a low-level (datalink layer) abstraction, important for performance, but provide all the functionality needed for sharing and management of network resources.

In a mobile environment, network resource management on a per-host basis is not adequate. The datalink layer must be able to manage resources on a per-connection basis. An example is the use of flow control for buffer space management. Per-host flow control is inadequate because multiple logical connections from a single host can compete destructively for resources allocated to that host. For this reason, per-connection flow control is needed. Similar arguments apply to other network management problems including priority, error correction, and error recovery mechanisms. In short, network management problems require the datalink layer to manage resources based on end-to-end connections. This resource management by end-to-end connections is functionally equivalent to the virtual circuit communications model. Making the virtual circuits explicit enables a cleaner and more efficient network implementation.

MOSS virtual circuits provide the following key functionality:

- They provide a connection between a client and a server at the datalink layer. This is in contrast to packet-oriented datalink services, which provide a connection between pairs of machines.
- They provide flow control, necessary for sharing of bandwidth between services [18].
- They provide flexible quality of service (QoS) guarantees. This makes it possible for a service to select the service level most appropriate for its needs [18].

MOSS will provide compatible virtual circuit implementations for a variety of communications media. In a mobile environment, where link bandwidth is limited and lossy links are common, these virtual circuits can respond rapidly and properly to fluctuations in available communication and computing resources. Furthermore, individual services can select appropriate QoS levels for their VCs, selecting performance criteria, such as reliability and security, that meet service requirements while optimizing the use of communications resources. Lastly, compatible virtual circuit implementations for various media will make it easier to change communications media without disturbing the service. Overall, MOSS virtual circuits

applied to mobile communications have many advantages over traditional networking models and provide a sound foundation upon which to build mobile services.

The second key abstraction in MOSS is the virtual device. MOSS virtual devices provide a convenient, flexible, and platform independent abstraction of remote resources. MOSS virtual devices have semantics similar to those provided by local devices. By providing device semantics instead of higher-level semantics, we keep the virtual device implementations simple, lightweight, and efficient. Virtual device implementations are fully aware that they are operating over a network. This makes it possible for them to apply device semantics to dynamic network resource management and error recovery, and protect applications from network idiosyncracies. The virtual device architecture consists of a client with which applications interact and a server that manages the physical device.

MOSS virtual devices benefit from the tight coupling of communication and service implementations. This tight coupling eliminates the overhead of protocol stacks, with services implemented as virtual devices directly on top of VCs. Services understand their own QoS requirements and can select a VC with the appropriate quality of service. For example, an audio virtual device would request a constant data rate connection and would “know” never to retransmit. In contrast, a virtual device providing access to file-system meta-data would require a fully reliable connection, but would be more tolerant of variations in transmission latency.

The ability to apply device semantics is particularly useful for error recovery. Rather than requiring successful transport of all packets, MOSS services can “fail up” to higher semantic levels. For example, when a media error occurs and a change of network media is required (e.g. from wired to wireless) a file-system virtual device can recover by reissuing pending disk requests — a straightforward procedure when compared to fully reliable packet transport.

In addition to virtual circuits and virtual devices, MOSS includes a Virtual Device Manager (VDM) and a set of services. The VDM provides the services required for maintenance and use of virtual devices (e.g. name management, network status information). MOSS services include the set of utilities that make virtual devices convenient to use.

The VDM is composed of four principal software components: the Directory Manager, the Virtual Device Agent Manager, the Network Manager, and the Virtual Circuit Manager. The *Directory Manager* provides a simple but flexible naming service for virtual devices. It implements a directory of devices provided locally, which can be queried both locally and over the network. An application can get a directory of the virtual devices provided by a specific host by querying that host’s Directory Manager through the local Directory Manager. On entering a new service area, a mobile host can query the local environment for information about local service providers.

The *Virtual Device Agent Manager* is responsible for starting VD servers and clients. When a virtual device becomes active, the Agent Managers on the server and client machines start their respective VD agents and provide them with a VC to establish the communication link between them. In “normal” operation, VD agents operate independently of the VDM once device service has begun; they send data and adjust link parameters without involving the Agent Manager. There are, however, some occasions where a VD agent will interact with the VDM:

- to share information about link quality,
- to receive notification from the VDM about a change in link quality, and
- to obtain a new VC when a complete link failure occurs.

These interactions are infrequent, and use asynchronous communications to avoid delays in service.

The VDM *Network Manager* is responsible for maintaining information about available network media and notifying VD agents about changes in link behavior as required by the virtual device. The *Virtual Circuit Manager* implements creation and destruction of virtual circuits. It provides VCs for newly activated virtual devices, and also after a media failure, when a virtual device must switch-over to a new physical medium.

Our prototype system will provide a core of MOSS services, such as:

- a local environment server,
- a conferencing system, and
- a maintenance system for mobile units.

Additional services will be created as the needs of mobile users are better understood.

MOSS requires system support beyond what is provided by conventional operating systems. It requires a fast path to the network interface and a mechanism for adapting system software to conform to changing properties in the communication media. To meet these requirements, we require an extensible operating system that allows user-level programs to modify the operating system's behavior. We will be developing this support in an experimental kernel, called VINO, that is being designed and developed at Harvard. The goals of VINO are threefold: to allow applications to specify the kernel policies that manage resources, to make kernel primitives accessible at user-level, and to provide a system interface designed around the acquisition of hardware and software resources. These three characteristics make VINO an ideal experimental platform for OS research in mobile systems and for implementation of MOSS.

The dynamically loaded libraries and loadable device drivers provided in current systems such as OSF/1, NetBSD, and Windows NT provide a crude approximation of the kind of support provided by extensible systems such as VINO. MOSS implementations for current systems will provide us with an exportable platform and permit us to decouple the operating system component of this work from the work related specifically to mobility.

The MOSS architecture is designed to permit mobile services to adapt to changes in the computing environment. With MOSS, mobile wireless services can change their usage of communications resources based on current network conditions, device semantics, and the availability of other resources such as power. The MOSS structure provides the essential mechanisms for mobile computations to share communications resources and make efficient use of available bandwidth.

3. Virtual Circuits

This section provides more technical details on MOSS virtual circuits. Virtual circuits provide a powerful abstraction for the implementation of network management. For example, flow-controlled VCs can prevent buffer overflow at the receiver. In a recent demonstration of our CreditNet experimental ATM network [5, 6, 13, 14, 15, 18] we showed that multiple TCP sessions competing for bandwidth over a single link through a switch with a 200-cell buffer achieve only 33 percent efficiency in the absence of flow control. With VC flow control, the efficiency rises to over 99 percent. When competing with high-volume UDP, a TCP session with ATM-level flow control can maintain its throughput even though the UDP session does not reduce its bandwidth during network congestion. In both cases, the improved performance is due to flow control on VCs; transient congestion from bursty UDP traffic no longer results in buffer overflow and dropped TCP packets.

A second key advantage of our approach is that it allows us to optimize each VC implementation for the properties of the underlying physical media. In this way we can provide a uniform QoS interface to virtual devices, while tuning QoS functionality (such as retransmissions) as appropriate for the physical media. Applications can specify link QoS levels to match their requirements, and the virtual circuits implementation accommodates the properties of the link. This is in contrast to current approaches that require augmenting higher level protocols or restricting the activity of applications to deal with the changes in behavior of the underlying media.

Our choice of the VC abstraction for wireless media is motivated by our experience with ATM (Asynchronous Transfer Mode) networks. VCs on ATM networks permit a low-level (datalink layer) implementation of functionality such as flow control, error recovery and network bandwidth management. Previous approaches required layered software implementations of high-level protocols with substantial overhead. As an illustration of the benefits of MOSS VCs on wireless media, consider the use of TCP for transmis-

sion of data over an unreliable radio link. As TCP cannot distinguish between a packet corrupted by the radio link and a packet lost due to congestion, it requires a time-out of one second or longer before retransmission. The penalties from timeouts are caused by TCP congestion window management. When radio transmission corrupts a packet, TCP assumes the packet was dropped due to congestion, shrinking its congestion window and drastically reducing its transmission rate. The result is that network performance seen by the application degrades precipitously. This theoretical phenomenon is a very real occurrence, and we observed it in the prototype mobile IP system we developed at Harvard last year [5].

To see the benefit of MOSS virtual circuits, note that the problems with TCP over a lossy wireless medium described earlier will not occur if TCP is used over a reliable MOSS VC datalink layer because retransmission due to lossy networks will be handled by the VC layer. The underlying VC guarantees reliable data delivery so that TCP will never see packet loss due to errors introduced by radio transmission media. This TCP example provides a simple illustration of the rationale behind our proposal. Typically, a virtual device will not use TCP. Instead of additional protocol layers, MOSS services can use VCs that implement the appropriate QoS directly.

A second example of the advantages of our approach is related to reliable VCs that can adaptively update the size of data packets based on the current bit error rate of the wireless medium. In the following we present a simple analysis to demonstrate the importance of adapting the size of data packets to the error rate when implementing a reliable virtual circuit over a lossy medium. Since this adaptation can be efficiently performed at the VC datalink layer, high-level services using these reliable VCs can benefit from improved performance, while being insulated from the low-level adaptive error handling scheme. This illustrates the power of the VC substrate concept.

Packet size selection becomes increasingly important as the bit error rate (BER) increases. Although wired networks have vanishingly small BERs, BERs in a radio environment can be as high as 0.01. An improper selection of packet size will cause a significant retransmission cost or high packet overhead. Consider the following analysis: Let H be the constant overhead in number of bits per packet, E the bit error rate, e the packet error rate, and P the packet size in bits (excluding overhead). We can derive the expected total transmission cost T in number of bits for delivering S bits of data as follows:

$$T = \frac{S}{P} (P + H) (1 + e + e^2 + e^3 + e^4 + \dots) = \frac{S}{P} (P + H) \frac{1}{1 - e}$$

Since e can be derived from bit error rate E , $e = 1 - (1 - E)^P$, we have:

$$T = \frac{\left(S + \frac{S \cdot H}{P} \right)}{(1 - E)^{(P + H)}}$$

Dividing T by S , we get the transmission cost, α , per bit:

$$\alpha = \frac{T}{S} = \frac{\left(1 + \frac{H}{P} \right)}{(1 - E)^{(P + H)}}$$

Note that $\alpha \geq 1$. For a given value of E , we are interested in choosing the proper value of P that minimizes α . Figure 1 depicts α as a function of packet size P under several values of error bit rate E . We see for example when $E = 0.001$ the optimal packet size P is about 80 bytes, and the α value increases rapidly

for larger values of P . If the packet size is not properly chosen, the retransmission overhead caused by errors can increase by two or three orders of magnitude.

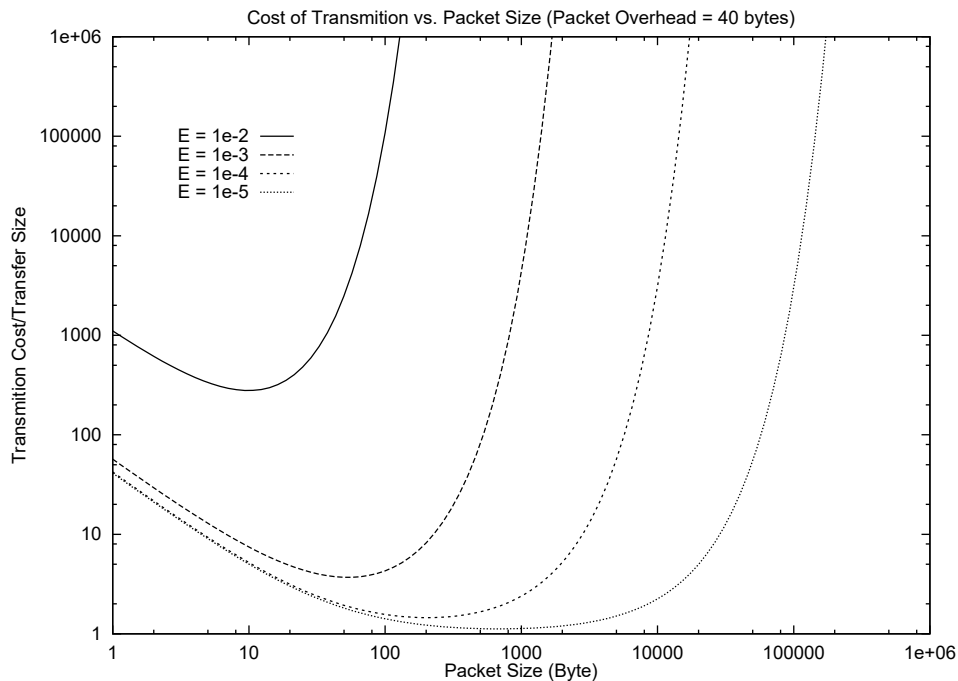


Figure 1. Value of Transmission Cost α (or T/S) as a Function of Packet Size

The optimal packet size for given values H and E is given by:

$$P = -\frac{H}{2} + \frac{\sqrt{H^2 - \frac{4 \cdot H}{\ln(1-E)}}}{2}$$

Figure 2 depicts the optimal packet size as a function of bit error rate for several packet overhead values. One of the goals in implementing a reliable VC is to adapt the packet size to the current bit error rate to maximize the utilization of wireless links.

Overall, the MOSS approach of using VCs of proper QoS to satisfy specific requirements of higher-level services is general and powerful. VCs are especially attractive for wireless networks as they provide a stable and efficient interface between the services and wireless media with inherently varying performance.

4. Virtual Devices

The two key properties of MOSS virtual devices are that they permit the integration of device semantics into communications management, and they protect applications from idiosyncracies of the network. Examples of virtual devices illustrate how the architecture will achieve its benefits in practice.

A *keyboard virtual device* requires a reliable virtual circuit with minimum communications latency and low throughput. On a lossy link with high latency the virtual device might choose to use forward error correction to avoid costly retransmissions.

A *disk virtual device* requires a reliable virtual circuit for communication of file system meta-data. The virtual circuit should have the best throughput available but with no special constraints for data rate or

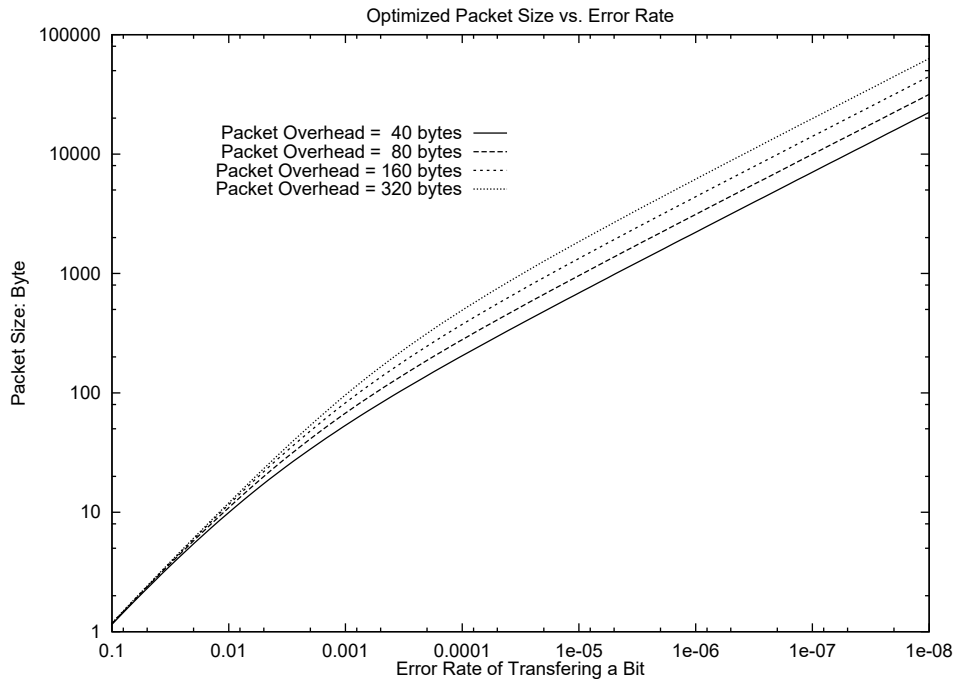


Figure 2. Optimal Packet Size as a Function of Bit Error Rate

latency. If CPU time were available the virtual disk device could choose to use compression on the network link, exercising the trade-off between computation and bandwidth.

The requirements of continuous media devices differ from those of more popular network applications because they generally require a constant data rate, and are often *error-tolerant* in the sense that they can use data with a small number of errors. A *screen display virtual device* uses a virtual circuit that does not correct transmission errors for pixel data. It might, however, use a second low-bandwidth reliable link for transmission of control information such as refresh and resize requests. An *interactive speech device* requires a virtual circuit with constant data rate, moderate bandwidth, and no error correction. A *low priority real-time video device* would use credit-based flow control for scheduling. The server would be scheduled to send the next frame when it had credit for a complete frame. Table 1 summarizes the quality of service needs for various virtual devices.

Virtual Device	Quality of Service Needs
Keyboard	Reliable, low bandwidth, low latency
Disk	Reliable, high bandwidth
Screen	Error-tolerant, reliable for control information
Audio	Error-tolerant, low latency
Video	Error-tolerant, high throughput

Table 1. Virtual Devices and Their Quality of Service Needs

One of the key technical challenges for MOSS virtual devices is dynamic adjustment of network parameters as media properties change. Two cases can be considered separately, the case where network service parameters change but the VC is still available, and the case of the complete failure of a connection. Consider the case where the current VC is still usable. A change in network parameters could be triggered by a change in the properties of the transmission media. For example, a VD agent using an unreliable

VC might configure the VDM to notify it if error rates exceed a certain threshold (so the packet size can be changed). Alternatively, a VD client might request a change in network parameters if it required a different service level due to application demands. As an example, a video display device that wanted to increase its resolution would request increased bandwidth from its VC. A change in network parameters requires a negotiation between client and server to agree on new parameters. The specific network parameters that might be negotiated between VD client and server depend on the virtual device. Some possibilities include:

- granularity and type of error detection codes
- type of error correction codes
- recovery protocol for uncorrectable errors
- use of compression
- use of encryption
- frame rate
- resolution
- change over to new physical network media
- packet size

The MOSS architecture gives virtual devices the freedom to implement their own VD specific communications options. When a VD agent decides it wants a change in communications parameters, it uses an in-band control channel provided with its VC to negotiate the change with the other VD agent. Each virtual device implements the options it needs, and the VC provides underlying support with selectable QoS.

An important benefit of the MOSS architecture is its straightforward approach to network error recovery. MOSS will support a powerful error recovery mechanism to handle the disconnections and service delays that are expected with wireless network media and mobility. VDs are responsible for recovery in the case of a complete failure of the physical media and subsequent change of media, and will use improved error recovery strategies that benefit from the integration of device semantics into error recovery. Some examples of virtual devices and the device-specific error recovery strategies they use are given in Table 2. Device-specific error recovery provides a software model in which virtual devices can reset themselves to a clean state and restart pending operations after a complete network failure. In the case of a media failure, MOSS devices that implement device-specific error recovery can switch over to a new virtual circuit on a different physical medium, assuming that a different medium is available.

Device	Error Recovery Mechanism
File system	Re-issue pending disk requests
Display	Repaint screen and continue
Keyboard	Clear input, signal user (^G), and continue
Mouse	Clear And Reset
Video	Continue from appropriate frame

Table 2. Device Specific Error Recovery

Implementing a Virtual Disk Device

In this section we illustrate how virtual devices provide access to remote resources by describing the implementation and use of a virtual device that provides access to a magnetic disk.

Disks are commonly accessed using a file system abstraction. In the following we describe the implementation and use of a raw disk device. Alternatively, a virtual device could provide disk access using a file system abstraction. The two interfaces have complementary advantages and disadvantages. The raw disk device permits a very simple, compact server implementation, while the file system device enables a very simple client. While the file system VD has better security properties, the raw disk will offer better performance in some situations. We plan to implement both virtual devices so that the best interface for a

given situation will be available. At the end of the section we discuss the advantages and limitations of the raw disk interface and the modifications necessary to support a file system virtual device.

A Raw Disk Virtual Device

Initial State

Assume the client wants the disk mounted whenever possible. The disk is mounted at boot time and made available whenever network connectivity is available. Both client and server have a VDM that knows about VD services provided locally and also knows how to contact VDMs on remote machines to access remote services.

Mounting the Disk

When the client machine is booted, the operating system “mounts” the virtual disk as described below.

- Using well-known names for the server and virtual disk, the client queries the server through its VDM.
- The client VDM creates a VD client. The server VDM starts a VD server if one does not already exist. The client and server VDMs create a VC and provide it to their respective VD agents.
- The client OS makes the disk accessible by updating its table of mounted file systems and by registering routines provided by the VD client that implement read and write operations on the raw disk.

Normal Operation with the Disk

An application issues a request to open a file on the file system corresponding to the mounted virtual disk. The operating system processes the request, passing through the software layer that implements the file system, which results in a series of read requests to the raw disk. The VD client processes each read request in turn, converting it into a message which is delivered on the VC to the VD server. The VD server responds with the requested data.

Change in Link Service Properties

Changes in link service properties are handled by the VD client and server. Communication takes place over the control channel provided by the VC. The application and operating system do not participate and will not be affected, except possibly by:

- changes in communications latency and throughput and
- changes in the amount of CPU time used by the VD agents.

Consider for example a VD client with ample power and CPU cycles but limited network bandwidth. The VD implements a simple mechanism by which the client asks the server to compress disk blocks before sending them over the network. This demonstrates how the MOSS architecture permits virtual devices to exercise trade-offs between network bandwidth, latency, CPU time, and power.

Link Failure

Link failure is a normal occurrence in mobile computing. Either the virtual circuit or the virtual device can determine that a given disk operation has failed. When an extreme change in network parameters (such as a complete loss of connectivity) occurs, the virtual circuit notifies the virtual device. In some cases connectivity may still exist but the bandwidth may be too low to make reasonable progress. When this occurs, the VD may use a timeout or a minimum throughput threshold set in the VC implementation to decide that bandwidth is too low to complete an operation within a reasonable delay.

When a link failure occurs the virtual device has several options.

- Use compression or device-specific methods to improve throughput.

- Ask the virtual circuit to increase power, request more bandwidth, etc.
- Report the problem to the user through a query box.
- Change network media; e.g., fail to wireless when Ethernet is disconnected.
- Become dormant until the link is available again.
- Report an error to the application.

The option is selected either by the file system running on top of the virtual device or through a dialog box provided by the virtual device client. Part of our project is to study mechanisms for handling media failure to understand how to provide the best service to the end user.

In the case of an extended link failure, the VD server cannot know if the client will be rebooted before the network becomes available again. The server might decide to shut down and deallocate the VC if it thinks the client will be gone for a long time. In this case the VD client might need to use device-specific error recovery when the connection becomes usable again, restarting the server and reissuing pending disk requests. Burden for error recovery is on the VD client.

Discussion

Three aspects of file system operation make virtual disk device service more complicated than most other virtual devices.

- File systems are commonly shared among multiple users.
- File systems must support multiple independent concurrent requests.
- Many operations that modify a file system require multiple updates to the raw disk. In the general case, the multiple updates must appear to be atomic.

Most MOSS virtual devices will implement a low-level abstraction that can be used by an application directly, despite being low-level. For a virtual device that provides access to a raw disk, this is not true. Because file systems are frequently shared, a significant amount of system functionality must be implemented on top of a raw disk on the disk client in order to support sharing.

A sophisticated raw disk VD implementation could provide some of the above functionality. For example, a logging or transaction protocol can be used to recover from errors that require an atomic operation to be aborted. The raw disk virtual device can implement access controls by permitting only authorized users/machines to access the physical device. It might further assume or require that authorized machines are running a version of the file system that implements access control on individual files. However, the security provided by such a system would not be adequate in all situations.

The alternative to the raw disk is the file system interface. It has the following advantages:

- Access controls are implemented by the server; there is no need to trust clients to implement access controls correctly.
- Atomicity is straightforward because the server serializes operations. This is beneficial for both sharing and error recovery.
- Open operations on files require only a single request on the network rather than one per raw disk request. In the absence of caching, each directory component of a path name requires two raw disk accesses.
- The file system VD accommodates a model in which each open file uses a different VC. This permits more fine-tuning of QoS levels.

Disadvantages of the file system interface are that it requires a more complicated server implementation and that the raw disk gives better performance in some situations. MOSS will provide the flexibility for implementing and evaluating both alternatives.

For either interface, caching will be an important factor in optimizing performance. A raw disk user would have the flexibility of caching raw disk blocks without software consistency guarantees. For the file

system virtual device there are many alternatives for supporting cache consistency among clients [7, 9, 11, 17, 19].

5. MOSS services

We are planning three MOSS services to validate our design. The first is the MOSS Local Environment Service (LES). It will make resources from the local environment available to mobile units for simple devices such as file systems, CD-ROM drives, keyboards, and displays. Through MOSS LES, many existing applications will benefit from the support for mobility provided by MOSS. MOSS LES will also serve as the basis for MOSS-specific applications.

The second major MOSS service is a group communication system. This service supports interactive video and audio, audio and video playback, and mechanisms for sharing applications, keyboards, and displays in an environment with multiple participants and requiring self-configuration. A typical use might be a briefing situation. Each members of the audience could make a portion of their PDA display available as a virtual device. The speaker's computer could access these devices to display data and images. By exporting a secure virtual-disk device, the audience members could permit the speaker to download documents and programs directly onto their machines. The speaker's system could also provide a collection of meeting notes and other documents for the audience members to download as desired.

The third service is an automated maintenance service for mobile units. The maintenance service will accommodate a computer usage model where mobile units require regular file system backups and software updates, but where periodic down-time or time away from their owner is not acceptable. In such a maintenance service, a maintenance agent on a mobile host uses local configuration to determine a maintenance schedule. A server on a stationary host provides virtual devices for maintenance services such as file backup media and software updates. VD clients for maintenance services are activated under appropriate conditions, e.g. the machine is idle, well connected, and the current time is between midnight and 6:00am. A virtual device on mobile hosts provides a database of PDA-specific information (e.g. owner, serial number, files backed up) to a stationary system that keeps maintenance records. Together, the three MOSS services will provide both useful functionality to mobile users, and a practical demonstration of the benefits of the structure of MOSS.

6. Related Work

The approach taken by most software projects in mobile computing is to build support for wireless networks on top of system software originally designed for wired networks. This approach is inherently flawed. The semantics of communication failures in mobile computing are different from those of wired systems, and operating systems and network protocols for wired computing do not properly address these failures. Mobile systems will only achieve good performance and robustness if mobility is inherently supported by the design of system software.

Several current projects at Carnegie Mellon University and elsewhere [2, 3, 10, 22] seek to adapt the Internet protocol suite for mobility. The multi-layer implementation employed by these systems makes it difficult to achieve rapid responsiveness, robustness and performance simultaneously. MOSS avoids this complexity by implementing system software directly on a VC layer. MOSS will achieve improved robustness and performance through a closer coupling of the communication substrate and service implementation.

The Wit project at the University of Washington seeks to make more efficient use of scarce wireless bandwidth through *application partitioning*, a model under which application functionality is split between the mobile client and the stationary server [25, 26]. The goal of application partitioning is to reduce bandwidth requirements by applying service-specific semantic information on the server side. Although this suggests some similarity between Wit and MOSS, the differences between the two approaches are substantial. MOSS considers services at a relatively low semantic level using a device-specific API. In contrast,

Wit operates at the semantic level of applications, using *hyperobjects* which export a Tcl API [21]. Wit assumes a standard network substrate, leaving problems of bandwidth sharing and dynamic packet size adjustment unsolved. Lastly, Wit places more of the burden of supporting mobility on application creators. In MOSS we protect applications by supporting mobility within virtual devices.

The Odyssey project at Carnegie Mellon University [20] proposes an ensemble of extensions to UNIX to support “application-aware adaptation” in mobile applications. They propose to augment current UNIX systems with

- a new set of system calls: The Odyssey API, and
- a new layer of software: Tomes.

By building a complex software layer on top of an already complex system, it is likely that the communication model Odyssey presents to mobile services will be slow and complex. MOSS avoids these problems. Its VC communication is simpler and more flexible than its legacy counterpart in Odyssey. With integration of a clean communications model into service implementation, the system will support mobility at all levels and present a cleaner interface to service implementations.

The goal of the InfoPad project at UC Berkeley [11, 16, 23] is to create an environment that will allow a large number of wireless computers in a confined area simultaneous access to and communication with multi-media network services. Their research covers an expansive range of topics, from circuit design for wireless CDMA [23] to hardware design for a mobile computer [24] to network design [16]. In contrast, MOSS will focus on communications and a software structure to support it. The InfoPad approach to wireless networking focuses on adding adaptation layers to IP protocols like TCP to provide reasonable behavior in wireless environments. In MOSS we avoid the complexity and performance penalties that occur when legacy protocols such as TCP are adapted for mobility.

Although the goals of the MosquitoNet [3] project are similar to ours, there are significant differences in the technical approach used to meet those goals. Where MOSS uses a virtual circuit layer as the foundation for mobile service implementations, MosquitoNet will interpose a network adaptation layer at the IP protocol level, while preserving current network software layers. The datalink approach used by MOSS provides increased flexibility and isolates the application from network adaptation. The MosquitoNet adaptable IP layer must manage local and remote connection information; in MOSS this information is managed transparently by changing the underlying VCs being used by a virtual device.

The mechanisms used by MOSS and Mosquito-Net to manage link bandwidth also differ. MOSS services specify quality of service once, when the application connects to the virtual device. The virtual device understands enough about semantics of the data to respond to changes in the underlying media without disturbing the application. In contrast, applications under MosquitoNet will have to respond to changes in network service characteristics as they occur. We believe that as more software layers are activated to deal with changes in the underlying physical network, increased complexity in application implementations as well as performance problems will result.

The tight integration of networking code and device implementation gives MOSS several advantages over the MosquitoNet approach. Tight coupling of network and device in MOSS makes fine-tuning of QoS possible. The software layers between MosquitoNet applications and the network implementation would make equivalent tuning awkward and complicated. MOSS virtual devices and virtual circuits also provide an architecture for implementing sharing of bandwidth on low-bandwidth links. There is no corresponding structure in MosquitoNet. Further, we expect that device specific error recovery in MOSS will be a powerful method for implementing efficient and robust mobile systems. There is no equivalent notion in MosquitoNet.

The VuNet Desk Area Network [1] provides access to intelligent networked devices over a wired ATM network. VuNet and MOSS both use a device interface and a virtual circuit-based network. As such VuNet’s functionality further validates our approach. However, VuNet differs from MOSS in that VuNet does not address mobility and makes heavy use of legacy software and protocols. In particular, VuNet communication is based on the BSD UNIX socket interface and TCP/IP. VuNet successfully supports

video on demand applications on high performance workstations (DEC Alpha 3000). An important property of video-on-demand is that, even though throughput requirements are high, so is tolerance for communication delays. Delay can, however, become a problem with latency-sensitive applications. MOSS addresses these issues with the use of priority VCs.

The design for our wireless VC communication model is inspired by the CreditNet experimental ATM network [5, 6, 13, 14, 15, 18]. CreditNet was used to demonstrate that credit-based flow control and QoS guarantees can be implemented directly at the VC level without the overhead of layered protocols.

7. Conclusions

Our goal in MOSS is to provide a sound systems framework for mobile computing. We hope to identify the key incompatibilities in current systems with respect to mobility and demonstrate a systems architecture that addresses the problems we identify.

MOSS virtual circuits provide a sound foundation for mobile data communications. By making end-to-end connections visible at the datalink layer, they simplify implementation of such functionality as bandwidth sharing, error recover, adaptive control of quality of service, functionality which is crucial to supporting advanced applications with the limited and varying bandwidth implied by wireless communication. Virtual circuits provide a framework with key functionality implemented in a media-dependent way while still providing a uniform communications interface to network clients. MOSS virtual circuits provide a vehicle for solving many of the problems in current communications systems with respect to mobility.

MOSS virtual devices make it possible to integrate device-specific semantics into communications resource management. This integration of device semantics into communications management makes it possible to apply powerful new techniques for error recovery in bandwidth management.

The effective use of wireless network resources in mobile computing depends on the ability of mobile systems to adapt to changes in the computing environment. Current protocols and systems are ill suited for mobility because they integrate the assumptions from two decades of static machine configurations and static network topologies. In MOSS we apply two abstractions, virtual devices and virtual circuits, to address the limitations of current systems. These abstractions permit MOSS services to adapt to changes in the environment and provide a sound foundation for mobile computing.

Bibliography

- [1] Adam, J.F, Houh, H.H., Ismert, M., and Tennenhouse, D.L., "A Network Architecture for Distributed Multimedia Systems", *Proceedings of the International Conference on Multimedia Computing and Systems*, IEEE Computing Society, May 1994.
- [2] Amir, E., Balakrishnan, H., Seshan, S., and Katz, R.H., Efficient TCP over Networks with Wireless Links. In Proc. HotOS-V, May 1995. To appear.
- [3] Baker, M., "Changing Communication Environments in MosquitoNet," *Proceedings of the 1994 IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.
- [4] Bershad, B.N., Chambers, C., Eggers, S., Maeda, C., McNamee, D., Pardyak, P., Savage, S., and Sirer, E.G. "SPIN - An Extensible Microkernel for Application-specific Operating System Services." *Proceedings of the 1994 European SIGOPS Workshop*, September 1994.
- [5] Blackwell, T. Chan, K., Chang, K., Charuhas, T., Gwertzman, T., Karp, B., Kung, H.T., Li, W.D., Lin, D., Morris, R., Polansky, R., Tang, D., Young, C., and Zao, J., "Secure Short-Cut Routing for Mobile IP," Conference Proceedings of Usenix Summer 1994 Technical Conference, Boston, Massachusetts, June 6-10, 1994, pp. 305-316. (The USENIX Association Best Student Paper Award)
- [6] Blackwell, T., Chan, K., Chang, K., Charuhas, T., Karp, B., Kung, H. T., Lin, D., Morris, R., Seltzer, M. Smith, M., Young, C., Bahgat, O., Chaar, M., Chapman, A., Depelteau, G., Grimble, K., Huang,

- S., Hung, P., Kemp, M., Mahna, I., McLaughlin, J., Ng, T., Vincent, J., and Watchor, J. "An Experimental Flow Controlled Multicast ATM Switch," *Proceedings of First Annual Conference on Telecommunications R&D in Massachusetts*, Vol. 6, pp. 33-38, October 25, 1994.
- [7] Dahlin, M., Mather, C., Wang, R., Anderson, T., and Patterson, D., "A Quantitative Analysis of Cache Policies for Scalable Network File Systems," *Proceedings of 1994 ACM SIGMETRICS*, Nashville TN, May 1994, pp. 150-160.
 - [8] Endo, Y., Gwertzman, J., Seltzer, M., Small, C., Smith, K., Tang, Diane "VINO: The 1994 Fall Harvest," Harvard University Technical Report TR-34-94.
 - [9] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., and West, M.J., "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, pp. 51-81.
 - [10] Johnson, D. "Scalable and Robust Internetwork Routine for Mobile Hosts," *Proceedings of the 14th International Conference on Distributed Computing Systems*, IEEE Computer Society, June 1994.
 - [11] Keeton, K. et al, "Providing Connection-Oriented Network Services to Mobile Hosts." *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, August 1993.
 - [12] Kistler, J.J., and Satyanarayanan, M., "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems*. Vol. 10, No. 1 (February 1992), pp. 3-25.
 - [13] Kung, H.T., Blackwell, T., and Chapman, A., "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing," *SIGCOMM '94*, pp. 101-114.
 - [14] Kung, H.T. and Chang, K., "Receiver-Oriented Adaptive Buffer Allocation in Credit-Based Flow Control for ATM Networks," *INFOCOM '95*, April 1995, pp. 239-252.
 - [15] Kung, H.T. and Morris, R., "Credit-Based Flow Control for ATM Networks," *IEEE Network Magazine*, Vol. 9, No. 2, March/April 1995, pp. 40-48.
 - [16] Le, M.T., Burghardt, F., Seshan, S., and Rabaey, J. "InfoNet: the Networking Infrastructure of InfoPad." *Proceedings of Comcon 1995*, San Francisco, CA, March 1995.
 - [17] Macklem, R., "Not Quite NFS, Soft Cache Consistency for NFS," *Proceedings of the 1994 Winter Usenix Technical Conference*, January 1994.
 - [18] Morris, R., and H. T. Kung, "Impact of ATM Flow Control on TCP Performance: Measurements on an Experimental ATM Switch," submitted to GLOBECOM'95.
 - [19] Nelson, M., Welch, B., Ousterhout, J., "Caching in the Sprite Network File System, *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988, 134-154.
 - [20] Noble, B., Price, M., and Satyanarayanan, M. "A Programming Interface for Application-Aware Adaptation in Mobile Computing," *Proceedings for the Second USENIX Symposium on Mobile and Location-Independent Computing*, April 1995.
 - [21] Ousterhout, J. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts. 1994
 - [22] Perkins, C. Myles, A., and Johnson, D., "The Internet Mobile Host Protocol (IMHP)." *Proceedings of INET '94, The Annual Conference of the Internet Society*, June 1994.
 - [23] Sheng, S., "A Monolithic CMOS Radio System for Wideband CDMA Communications." *Calgary 94 Wireless Conference*, 1994.
 - [24] Truman, T., "Infopad: A System Design for Portable Multimedia Access." *Calgary Wireless 94 Conference*, 1994.

- [25] Watson, T., "Effective Wireless Communication through Application Partitioning," Fifth Workshop on Hot Topics in Operating Systems, Orcas Island, WA, May 1995.
- [26] Watson, T., "Wit: An Infrastructure for Wireless Palmtop Computing," Technical Report UW-CSE-94-11-08, University of Washington, November 1994.