MobiDesk: Mobile Virtual Desktop Computing

Ricardo Baratto Shaya Potter Gong Su Jason Nieh
Department of Computer Science
Columbia University
{ricardo, spotter, gongsu, nieh}@cs.columbia.edu

Columbia University Technical Report CUCS-014-04, March 2004

Abstract

We present MobiDesk, a mobile virtual desktop computing hosting infrastructure that leverages continued improvements in network speed, cost, and ubiquity to address the complexity, cost, and mobility limitations of today's personal computing infrastructure. MobiDesk transparently virtualizes a user's computing session by abstracting underlying system resources in three key areas: display, operating system and network. MobiDesk provides a thin virtualization layer that decouples a user's computing session from any particular end user device and moves all application logic from end user devices to hosting providers. MobiDesk virtualization decouples a user's computing session from the underlying operating system and server instance, enabling high availability service by transparently migrating sessions from one server to another during server maintenance or upgrades. We have implemented a MobiDesk prototype in Linux that works with existing unmodified applications and operating system kernels. Our experimental results demonstrate that MobiDesk has very low virtualization overhead, can provide a full-featured desktop experience including full-motion video support, and is able to migrate users' sessions efficiently and reliably for high availability, while maintaining existing network connections.

1 Introduction

Continuing advances in hardware technology have enabled the proliferation of faster, cheaper, and more portable personal computers to support increasingly mobile users. As personal computers become more ubiquitous in large corporate and academic organizations, the total cost of owning and maintaining them is becoming unmanageable. These computers are increasingly networked, which only complicates the management problem with the need to constantly upgrade and patch to protect them and their data against a myriad of viruses and other attacks propagated through the network. Furthermore, as mobile users transport their portable computers from one place to another, it is not uncommon for these machines to be damaged or stolen, resulting in the loss of any important data stored on those machines. Even in the best case when such data can be recovered from backup, the time consuming process of reconstituting the state of the lost machine on another device results in a huge disruption in critical computing service for the user.

We introduce MobiDesk, a mobile virtual desktop computing hosting infrastructure that leverages rapid improvements in network bandwidth, cost, and ubiquity to address the limitations of the current personal computing model. With wire-speed network technologies scaling at faster Moore's exponents than silicon, MobiDesk leverages the network to decouple a user's desktop computing session from the end user device by moving all application logic from end user devices to hosting providers. End user devices are simply used for sending user input to the hosting provider and displaying output from the hosting provider, allowing them to be simple stateless clients. Input and output associated with a computing session can be easily redirected to any end user device. MobiDesk also decouples a user's desktop computing session from the underlying operating system and server instance, allowing a user's entire computing environment to be migrated transparently from one server to another. This enables a server to be brought down for maintenance and upgraded in a timely manner with minimal impact on the availability of a user's computing services. Once the original machine has been updated, the user's computing session can be migrated back and continue to execute even though the underlying operating system may have changed. MobiDesk ensures that any network connections associated with the user's computing session are maintained even as the session is migrated from one machine to another. MobiDesk provides these benefits without modifying, recompiling, or relinking applications or operating system kernels. MobiDesk requires no changes to clients other than being able to execute a simple user-space application to process and display input and output.

MobiDesk provides a mobile virtual desktop computing environment by introducing a thin virtualization layer between a user's computing environment and the underlying system. MobiDesk focuses on virtualizing three key system resources, display, operating system, and network resources. MobiDesk virtualizes display resources by providing a virtual display driver interface and framebuffer that efficiently encodes and redirects display updates from the server to an end user device. MobiDesk virtualizes operating system resources by providing a virtual private namespace for each desktop computing session that offers a host-independent virtualized view of an operating system, enabling the session to be transparently migrated from one server to another. MobiDesk virtualizes network resources by providing virtual address identifiers for connections and a transport-independent proxy mechanism to preserve all network connections associated with a user's computing session even it if is migrated from one server to another inside the MobiDesk server infrastructure.

The MobiDesk hosted desktop computing approach provides a number of important benefits over current computing approaches:

- *Independence from desktop failures*: Because application state and user computing environments are maintained by MobiDesk in a hosting provider, users are isolated from local desktop failures. There is no localized computing state that needs to be recovered. Furthermore, such devices can be inexpensive stateless machines that never need to be backed up or restored.
- *Transparent user mobility*: Because all persistent user state is maintained on the servers, users are able to move freely among any client access devices and pick up right where they left off.
- *Global computing access*: As a result of the widespread proliferation of Internet-connected devices, MobiDesk users can access their desktop computing environments from around the world on whatever computing infrastructure happens to be available.
- *High availability application services in the presence of server downtime and upgrades*: Because applications are decoupled from the server hardware and underlying operating system instance, applications can be moved anywhere and in particular migrated off faulty hosts, and before host maintenance and upgrades, so applications continue to run with minimal downtime. Today, maintaining and upgrading a server can result in long periods of service downtime, whereas MobiDesk enables hardware and operating systems to be upgraded in a timely manner with minimal impact on application service availability – by migrating applications to another machine that has already been updated. With MobiDesk, system administrators no longer need to schedule downtime in advance and in cooperation with all the users, thereby closing the vulnerability window of unrepaired systems. Once the original machine has been updated, applications can be migrated back, if appropriate.
- *Reliable computing for legacy applications and commodity systems*: Because MobiDesk is designed to work with unmodified legacy applications and commodity operating systems, it offers the potential to bring about more reliable computing without giving up the large investments already made in the existing software base.
- *Persistence and continuity of business logic*: By moving away from the current model of simply backing up file data to secure remote locations, and instead protecting entire computing environments by running hosting providers in secure remote locations, MobiDesk can enable academic, business, and government institutions to function much more effectively in times of crisis. Restoring an organization's local computing infrastructure from backup consequent to a crisis is an extremely slow, time-consuming process that is increasingly ineffective given the scale of IT infrastructure being deployed today. MobiDesk offers a different, improved model of continuous uptime, especially during a crisis, when infrastructure availability is most crucial.
- *On-demand access to scalable computational resources beyond the desktop*: By multiplexing a large pool of shared computational resources among many users, an individual can gain access to substantially more resources than can be afforded on one's local desktop computer, analogous to "grid computing" but in a technologically simpler fashion for the average user to exploit. The resources given to a user can be scaled up or down as necessary. Instead of having to throw away their existing local desktop machines every time they need more compute power, users just ask their service provider to scale up their resource allocation. More importantly, such scalable computing power is available on demand and can be allocated immediately even in times of crisis when computing resources may be most needed.
- *Makes available a wider range of application services*: Given the cost of computer software, many users often cannot afford more than a few, heavily used applications on their desktops. MobiDesk can provide a wider range of affordable application services on multiple operating system platforms by amortizing the costs of applications over a large number of users. Since not all applications will be in use by all users at one time, statistical multiplexing can serve a larger number of users with fewer software licenses. Just as MP3 download services have enabled less-known musicians to distribute their music, MobiDesk enables a greater diversity of software developers to provide innovative soft-

ware and distribute it to users via the computing service provider – potentially leading to new business models that benefit software developer, service provider and consumer alike.

- *Support for secure, low-cost client access devices*: Client access devices just need to be able to connect to the Internet. They do not need to provide complex computing functionality, making it unnecessary to continuously upgrade to more powerful desktop machines. Simpler, lower cost, possibly longer battery-life client access devices can be made more readily available for such a service. These devices may come in many shapes and sizes, from desktop machines with megapixel displays to handheld devices with pocket-sized screens. Furthermore, MobiDesk provides a model that can more easily secure low-cost client devices since they do not store persistent, sensitive data.

- *Bridging the information gap*: Contrasting with the current society of information climate "haves" and "have-nots," MobiDesk can be accessed by low-cost client devices to deliver secure service on a subscription basis offering a societally better way of providing access to computing as widely as possible. Remote computing and storage services can be paid for using a subscription-based or use-based economic model that follows current Internet access or telephone pricing policies. A MobiDesk provider could supply basic computing, including a complete suite of desktop productivity tools, for just a few dollars a month. Video-on-demand service, online gaming, and other multimedia services could be obtained from the same or other providers for additional small fees. Furthermore, the more centralized service and management model provides lower total cost of ownership across all applications, making computing more widely affordable. More importantly, MobiDesk enables client devices to be easier to manage and use by removing the local system complexity associated with current software upgrade cycles, making computing more accessible to a wider population of users.

This paper presents the design and implementation of MobiDesk. Section 2 briefly presents the overall MobiDesk system architecture and usage model. Section 3 describes the MobiDesk display virtualization mechanism. Section 4 describes the MobiDesk operating system virtualization mechanism. Section 5 describes the MobiDesk network virtualization mechanism. Section 6 presents experimental results measuring MobiDesk system performance and associated overhead in the context of real desktop computing applications. Section 7 discusses related work. Finally, we present some concluding remarks.

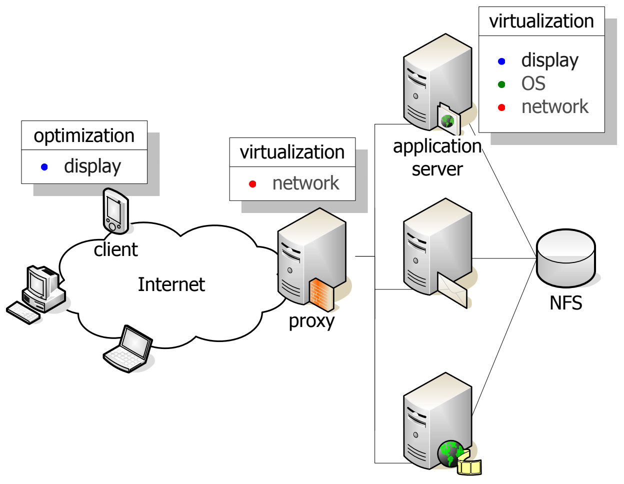

Figure 1: MobiDesk architecture

2 Overview of System Architecture

MobiDesk is architected as a proxy-based server cluster system, comparable to systems deployed today by application service providers. The overall architecture of the system is depicted in Figure 1. MobiDesk is composed of a proxy, a group of backend session servers connected in a LAN, a storage server infrastructure, and a number of external, heterogeneous clients through which users access the system.

The proxy acts as a front-end that admits service requests from clients across the Internet, and dispatches the requests to the appropriate backend application servers. The proxy exposes a single entry point to the clients, operates at layer 7, and employs suitable admission and service dispatching policies. The backend compute servers host completely virtualized environments within which the computing sessions of MobiDesk's users run. The network storage server infrastructure is used for all persistent file storage. The clients are merely *inputing* and *outputing* devices connected to the servers across the Internet.

Users interact with their MobiDesk sessions through a thin-client style *session viewer*, a simple device or application that relays the user's input and the session's output between the client and the server through a secure encrypted channel. Each user in the system is assigned a username and password. When a user first logs into MobiDesk, the proxy performs appropriate authentication, and connects her to a MobiDesk session server. The server creates a virtual and private environment, that is populated with a complete set of operating system resources and desktop applications. To the user, the session appears no different than to logging into a private computer. When the client disconnects, the session continues to run on the MobiDesk server, unless the user explicitly logs out. On future connections, the session will be in the same state it was when the user last disconnected from

MobiDesk.

By providing a virtual, private environment for each user, MobiDesk is able to dynamically relocate sessions to meet load balancing, system maintenance and/or quality of service requirements. Sessions can be checkpointed and migrated transparently at any point in time. To keep track of the sessions as migration occurs, MobiDesk implements a *session cookie* mechanism. As new sessions are created, the proxy generates a unique cookie that is passed to the hosting servers and associated with the new session. Whenever a session is migrated, the destination server uses the cookie to inform the proxy of the new location. Finally, the next time the user logs in, the proxy will use the cookie to identify the server where the user's session is being hosted.

To provide a private, mobile environment for user sessions, MobiDesk virtualizes three key resources: display, operating system, and network. MobiDesk virtualization is designed to work with existing unmodified applications, operating system kernels, and network infrastructure and protocols. The three components work in concert to create a completely virtualized environment for client computing sessions.

MobiDesk virtualizes the server display by providing a virtual display driver that intercepts drawing commands from user's applications, and translates the commands into a display protocol between the client and the server. MobiDesk display virtualization focuses on the importance of latency-sensitive design for interactive computing and anticipates the kind of high-bandwidth network access that is becoming increasingly cost-effective and accessible in WAN environments [2]. For example, South Korea is building a nationwide Internet access infrastructure to make speeds up to 100 Mbps available to the home by 2010 [17].

MobiDesk operating system virtualization provides a virtual, private namespace for each hosted client computing session. For example, each computing session has its own host-independent view of a complete set of OS resources such as PID/GID, IPC, memory, file system, and devices, etc. MobiDesk virtualization operates at a finer granularity than virtual machine approaches such as VMware [39] by virtualizing individual computing sessions instead of complete operating system environments. As a result, computing sessions can be decoupled from the underlying operating system and migrated to other servers to maintain high availability even in the presence of server hardware and operating system maintenance and upgrades.

MobiDesk network virtualization provides persistent network connections for client computing sessions even as they move among servers in a MobiDesk cluster. All connections operate through the MobiDesk proxy. Similar to operating system virtualization, MobiDesk virtualizes network connections by a virtual and private namespace for transport connection identifiers, such as IP address and port number, on both the proxy and the servers. These virtual identifiers remain constant and are simply translated to the underlying physical network identifiers as a session moves amongst servers at different physical network locations. MobiDesk network virtualization provides persistent connections from mobile client computing sessions to outside hosts without running any software on the outside hosts and without any changes to the existing network infrastructure.

3 Display Virtualization

To make MobiDesk a viable replacement to the traditional desktop computing model, MobiDesk needs to be able to deliver the complete set of unmodified desktop applications to end users with good performance. MobiDesk must work within the framework of existing display systems, intercepting display commands from unmodified applications and redirecting these commands to remote clients. To provide good WAN performance, MobiDesk must intercept display commands at an appropriate abstraction layer to provide sufficient information to optimize the processing of display commands in a latency-sensitive manner. Furthermore, to support transparent user mobility and eliminate client administration complexity, MobiDesk should support the use of thin, stateless clients by ensuring that all persistent display state is stored in the MobiDesk server infrastructure.

Given that traditional display systems are structured in multiple abstraction layers, there are a number of possible ways in which MobiDesk can interact with existing display systems. We can loosely categorize display system structure into three layers: application, middle-ware, and hardware. The application layer is the top display system layer with which applications interact. It presents a high-level model of the overall characteristics of the display system and includes descriptions of the operation and management of windows, input mechanisms, and display capabilities of the system. These capabilities may range from basic 2D display, to complex operations involving transparency, blends, 3D transformations and the display of multimedia content. The middle-ware layer sits between the high-level application display layer and the low-level video hardware layer. Its responsibility is to create a hardware-independent abstraction of the display hardware to meet the requirements of the display system and its applications. To maintain consistency across hardware with differing abilities, it is provisioned with fallback mechanisms and software routines that can implement missing hardware features. The video hardware layer is a a low-level, hardware-dependent layer that exposes the video hardware to the display system. It is implemented as a set of device drivers responsible for translating between the middle-ware's abstract display operations and the commands understood by the display hardware.

MobiDesk rules out using the application and middle-ware layers for intercepting display commands for the following important reasons. MobiDesk does not intercept at the application layer because that requires a significant amount of application logic and computational power on the client for translating high-level commands. This would limit the range of MobiDesk's target client architectures. Intercepting at the application layer also results in direct synchronization between applications and the client, which can reduce display performance in higher latency WAN environments. MobiDesk also does not intercept at the middle-ware layer because that would require MobiDesk to reimplement substantial display system functionality instead of leveraging continuing advances in existing middle-ware implementations such as XFree86.

MobiDesk display virtualization is instead designed as a virtual video device driver that intercepts display commands at the video hardware layer. MobiDesk provides a separate virtual video device for each computing session. Rather than sending display commands to local display hardware, MobiDesk's virtual video driver packages up display commands associated with a user's computing session and sends them over the network to a remote client. For this purpose, MobiDesk implements a simple, low-level, minimum overhead protocol, as described in Table 1. The protocol mimics the operations most commonly found in display hardware, allowing clients to do little more than forward protocol commands to their local video hardware, reducing the latency of display processing. To provide security, all protocol traffic is encrypted using the standard RC4 [33] stream-cipher algorithm. MobiDesk's video hardware layer approach allows it to take full advantage of existing infrastructure and hardware interfaces, while maximizing client resources and requiring minimal computation on the client. Furthermore, new video hardware features can be supported with at most the same amount of work necessary for supporting them in traditional desktop display drivers. While there is some loss of semantic display information at the low level video device driver interface, our experiments with desktop applications such as web browsers indicate that the vast majority of application display commands issued can map directly to standard video hardware primitives. Furthermore, we show in Section 6 that the simpler display virtualization used by MobiDesk can provide superior display performance compared to other approaches.

To deliver good performance in WAN environments, the MobiDesk display virtualization architecture couples its virtual video device driver approach with other latency-sensitive display mechanisms. In particular, MobiDesk exports client display hardware resources to the server and leverages such resources to reduce the latency of display processing. For example, MobiDesk provides direct video support by leveraging alternative YUV video formats natively supported by almost all off-the-shelf video card available today. Video data can be simply transferred from server to client video hardware, which automatically does inexpensive, high-speed, color-space conversion and scaling. As another example, MobiDesk leverages cursor drawing support available with almost every video card in use today. MobiDesk uses local cursor drawing in response to mouse movements and maintains local cursor state at the client. Cursor changes still come from the server, but MobiDesk improves system response time by avoiding network latency with any cursor drawing that does not result in cursor changes.

Command	**Description**
RAW	Display raw pixel data at a given location and size
COPY	Copy frame buffer area to specified coordinates
SFILL	Fill an area with a given pixel color value
BITMAP	Fill a region using a bitmap image
PFILL	Tile a pixmap rectangle in a given region

Table 1: MobiDesk Protocol Display Commands

MobiDesk provides two important server-side mechanisms for improving performance in WAN environments. One MobiDesk mechanism is the use of a server-push model for sending display updates to the client. As soon as display updates are generated on the server, they are delivered to the client. MobiDesk does not require clients to explicitly request display updates, which add additional network latency to command processing. Another MobiDesk mechanism is the use of display command scheduling to improve the responsiveness of the system. MobiDesk display virtualization employs a *Shortest-Remaining-Size-First (SRSF)* preemptive command scheduler, that is analogous to Shortest-Remaining-Processing-Time (SRPT). SRPT is known to be optimal for minimizing mean response time [5], a primary goal for an interactive system. In display applications, short jobs are normally associated with text and general GUI layout components, which are critical to the usability of the system. On the other hand, large jobs are normally lower priority "beautifying" GUI elements, such as image decorations, desktop backgrounds and web page banners.

Finally, MobiDesk supports thin, stateless display clients by storing at all times, all session state at the respective MobiDesk server. Although the MobiDesk takes advantage of client resources when available, all client state used by MobiDesk is considered temporary and destroyed upon client disconnect. When a remote client connects to the MobiDesk infrastructure, the server running the user's computing session transfers the current session state to the client. For the duration of the connection, the client forwards input events to the server, which in turn forwards display updates back to the client. The client at no point has an intermediate session state differing from the server. When the client eventually disconnects, it leaves no state behind in the local computer.

4 Operating System Virtualization

To enable users' computing sessions to freely migrate from one MobiDesk server to another, MobiDesk encapsulates each session with a host-independent virtualized view of the operating system. MobiDesk operating system virtualization provides the same application interface as the underlying operating system so that legacy applications can execute in the context of a session without any modification. Processes within a session can make use of all available operating system services, just like processes executing in a traditional operating system environment. Unlike a traditional operating system, each virtualized session provides a self-contained unit that can be isolated from the system, checkpointed to secondary storage, migrated to another machine, and transparently restarted. This is made possible because each computing session has its own private, virtual namespace. All operating system resources are only accessible to processes running in a session through the session's private, virtual namespace.

A session namespace is private in that only processes within the session can see the namespace. It is private in that it masks out resources that are not contained within the session, including processes outside of the session. Processes inside a session appear to one another as normal processes that can communicate using traditional IPC mechanisms. Other processes outside a session do not appear in the namespace and are therefore not able to interact with processes inside a session using IPC mechanisms such as shared memory and signals. Instead, processes outside the session can only interact with processes inside the session using normal RPC mechanism that support process communication across machines.

A session's namespace is virtual in that all operating system resources including processes, user information, files, and devices are accessed through virtual identifiers within a session. These virtual identifiers are distinct from host-dependent resource identifiers used by the operating system. The session's virtual namespace provides a host-independent view of the system by using virtual identifiers that remain consistent throughout the life of a process in the session, regardless of whether the session moves from one system to another. Since the session's namespace is separate from the underlying operating system namespace, the session's namespace can preserve this naming consistency for its processes even if the underlying operating system namespace changes, as may be the case in migrating processes from one machine to another.

The session's private, virtual namespace enables processes running in a session to migrate together as a group. Processes in a session have a consistent, host-independent view of the underlying operating system. Operating system resource identifiers such as process IDs (PIDs) must remain constant throughout the life of a process to ensure its correct operation. However, when a process is moved from one operating system to another, there is no guarantee that the underlying operating system will provide the same identifiers to a migrated process; those identifiers may in fact already be used by other processes in the system. The session's namespace addresses these issues by providing consistent, virtual resource names in place of host-dependent resource names such as PIDs. Names within a session are trivially assigned in a unique manner in the same way that traditional operating systems assign names, but such names are localized to the session. Since the namespace is private to a given session, there are no resource naming conflicts for processes in different sessions. There is no need for the session's namespace to change when the session is migrated, which allows sessions to ensure that identifiers remain constant throughout the life of the process, as required by legacy applications that use such identifiers. Similarly, the private virtual namespace enables sessions to be securely isolated from each other by providing complete mediation to all operating system resources. Since the only resources within each session are the ones that are accessible to a particular user, including his files and processes, if another user's session would get exploited, it would be unable to harm any other user's session.

4.1 Session Virtualization

Sessions are supported using virtualization mechanisms that translate between the session's virtual resource identifiers and operating system resource identifiers. Every resource that a process in a session accesses is through a *virtual name* which corresponds to an operating system resource identified by a *physical name*. When an operating system resource is created for a process in a session, such as with process or IPC key creation, instead of returning the corresponding physical name to the process, the session's virtualization layer catches the physical name value, and returns a private virtual name to the process. Similarly, any time a process passes a virtual name to the operating system, the virtualization layer catches it and replaces it with the appropriate physical name. The key virtualization mechanisms used are a system call interposition mechanism and the `chroot` utility with file system stacking for file system resources.

Session virtualization employs system call interposition to wrap existing system calls to check and replace arguments that take virtual names with the corresponding physical names before calling the underlying original system call. Similarly, the wrapper is used to capture physical name identifiers that the original system calls return and return corresponding virtual names to the calling process running inside the session. Session virtual names are maintained consistently as a session migrates from one machine to another and are remapped appropriately to underlying physical names that may change as a result of migration. Session system call interposition also masks out processes inside of a session from processes outside of the session to remove any interprocess host dependencies across the session boundary. System call interposition is used to virtualize operating system resources including process identifiers, keys and identifiers for IPC mechanisms such as semaphores, shared memory, and message queues, and network addresses.

Session virtualization employs the `chroot` utility and file systems stacking to provide each session with its own file system namespace that can be separate from the regular host file system. The session file system can be composed from loopback mounts from the host for sessions that are only checkpointed and restarted on the same machine. Similarly, one can make use of a portable hard drive that one moves between the different hosts one wants to migrate within. More commonly, the session's file system is composed from remote mounts via a network file system such as NFS so that the same files can be made consistently available as a session is migrated from one machine to another. More specifically, when a session is created or moved to a host, a private directory named according to the session identifier is created on the host to serve as a staging area for the session's virtual file system. Within this directory, the various network-accessible directories that the session is configured to access will be mounted from a network file server. For example, from a Unix-centric viewpoint, this set of directories could include `/etc`, `/lib`, `/bin`, `/usr`, and `/tmp`. The `chroot` system call is then used to set the staging area as the root directory for the session, thereby achieving file system virtualization with negligible performance overhead. This method of file system virtualization provides an easy way to restrict access to files and devices from within a session. This can be done by simply not including file hierarchies and devices within the session's file system namespace. If files and devices are not mounted within the session's virtual file system, they are not accessible to the session's processes.

Because commodity operating systems are not built to support multiple namespaces, a security issue that session virtualization must address is that there are many ways to break out of a standard chrooted environment, especially if one allows the `chroot` system call to be used by processes in a session. The session's file system virtualization enforces the chrooted environment and ensures that the session's file system is only accessible to processes within the given session by using a simple form of file system stacking to implement a barrier. This barrier directory prevents processes within the session from traversing it. Since the processes can't traverse the directory, they can't access files outside of the session's file system namespace.

In order for a session to fully replace a regular computer, the session has to allow processes to run as the privileged root user. For instance, programs such as `ping` and `traceroute` that need to create raw sockets and `passwd` that is used to change system resources need to run with privilege. Because the root's UID 0 is treated specially by the operating system kernel, session virtualization also treats UID 0 processes inside of a session in a special way to prevent them from breaking the session abstraction, accessing resources outside of the session, and causing harm to the host system. While a session can be configured for administrative reasons to allow full privileged access to the underlying system, we focus on the case of sessions for running application services which do not need to be used in this manner. Session do not disallow UID 0 processes, which would limit the range of application and services that could be run inside a session. Instead, sessions provide restrictions on such processes to ensure that they function correctly inside of a session.

While a process is running in user space, the UID it runs as doesn't have any effect. Its UID only matters when it tries to access the underlying kernel via one of the kernel entry points, namely devices and system calls. Since a session already provides a virtual file system that includes a virtual `/dev` with a limited set of secure devices, the device entry point is already secured. The only system calls of concern are those that could allow a root process to break the session abstraction. Only a small number of system calls can be used for this purpose. Session virtualization classifies these system calls into three classes that need to be protected.

The first class of system calls are those that only affect the host system and serve no purpose within a session. Examples of these system calls include those that load and unload kernel modules or that reboot the host system. Since these system calls only affect the host, they would break the session security abstraction by allowing processes within it to make system administrative changes to the host. System calls that are part of this class are therefore made inaccessible by default to processes running within a session.

The second class of system calls are those that are forced to run unprivileged. Just like NFS, by default, squashes root on a client machine to act as user `nobody`, session virtualization forces privileged processes to act as the `nobody` user when it wants to make use of some system calls. Examples of these system calls include those that set resource limits and `ioctl` system calls. Since system calls such as `setrlimit` and `nice` can allow a privileged process to increase its resource limits beyond predefined limits imposed on session processes, privileged processes are by default treated as unprivileged when executing these system calls within a session. Similarly, the `ioctl` system call is a system call multiplexer that allows any driver on the host to effectively install its own set of system calls. Since the ability to audit the large set of possible system calls is impossible given that sessions may be deployed on a wide range of machine configurations, session virtualization conservatively treats access to this system call as unprivileged by default.

The final class of system calls are calls that are required for regular applications to run, but have options that will give the processes access to underlying host resources, breaking the session abstraction. Since these system calls are required by applications, the session checks all their options to ensure that they are limited to resources that the session has access to, making sure they aren't used in a manner that breaks the session abstraction. For example, the `mknod` system call can be used by privileged processes to make named pipes or files in certain application services. It is therefore desirable to make it available for use within a session. However, it can also be used to create device nodes that provide access to the underlying host resources. To limit how the system call is used, the session system call interposition mechanism checks the options of the system call and only allows it to continue if it's not trying to create a device.

4.2 Session Migration

To maintain user computing session availability even in the presence of server downtime due to hardware and operating system upgrades, MobiDesk provide a checkpoint-restart mechanism that allows sessions to be migrated across machines running different operating system kernels. Upon completion of the upgrade process, the respective MobiDesk session and its applications can be restored on the original machine now with an upgraded operating system. We assume here that the systems have not been compromised and that any kernel security holes on the unpatched system have not yet been exploited on the system; migrating across kernels that have already been compromised is beyond the scope of this paper.

We also limit our focus to migrating between machines with a common CPU architecture with kernel differences that are limited to maintenance and security patches. These patches often correspond to changes in the minor version number of the kernel. For example, the Linux 2.4 kernel has more than twenty minor versions. Even within minor version changes, there can be significant changes in kernel code. For example, comparing across different Linux 2.4 kernel versions, all of the files for the VM subsystem were changed at some point since extensive modifications were made to implement a completely new page replacement mechanism in Linux. Many of the Linux kernel patches contain security vulnerability fixes, which are typically not separated out from other maintenance patches. We similarly limit our focus to scenarios where the application's execution semantics, such as how threads are implemented and how dynamic linking is done, do not change. On the Linux kernels this is not an issue as all these semantics are enforced by user-space libraries. Whether one uses kernel or user threads, or one how libraries are dynamically linked into a process is all determined by the respective libraries on the file system. Since the session has access to the same file system on whatever machine it is running on, these semantics stay the same.

To support migration across different kernels, MobiDesk use a checkpoint-restart mechanism that employs an intermediate format to represent the state that needs to be saved on checkpoint. On checkpoint, the intermediate format representation is saved and digitally signed to enable the restart process to verify the integrity of the image. Although the internal state that the kernel maintains on behalf of processes can be different across different kernels, the high-level properties of the process are much less likely to change. We capture the state of a process in terms of higher-level semantic information specified in the intermediate format rather than kernel specific data in native format to keep the format portable across different kernels. For example, the state associated with a Unix socket connection consists of the directory entry of the Unix socket file, its superblock information, a hash key, and so on. It may be possible to save all of this state in this form and successfully restore on a different machine running the same kernel. But this representation of a Unix socket connection state is of limited portability across different kernels. A different high-level representation consisting of a four tuple, virtual source pid, source fd, virtual destination pid, destination fd is highly portable. This is because the semantics of a process identifier and a file descriptor is typically standard across different kernels, especially across minor version differences.

The intermediate representation format used by MobiDesk for migration is chosen such that it offers the degree of portability needed for migrating between different kernel minor versions. If the representation of state is too high-level, the checkpoint-restart mechanism could become complicated and impose additional overhead. For example, the MobiDesk system saves the address space of a process in terms of discrete memory regions called VM areas. As an alternative, it may be possible to save the contents of a process's address space and denote the characteristics of various portions of it in more abstract terms. However, this would call for an unnecessarily complicated interpretation scheme and make the implementation inefficient. The VM area abstraction is standard across major Linux kernel revisions. MobiDesk view the VM area abstraction as offering sufficient portability in part because the organization of a process's address space in this manner has been standard across all Linux kernels and has never been changed since its inception.

MobiDesk further support migration across different kernels by leveraging higher-level native kernel services to transform intermediate representation of the checkpointed image into an internal representation suitable for the target kernel. Continuing with the previous example, MobiDesk restore a Unix socket connection using high-level kernel functions as follows. First, two new processes are created with virtual PIDs as specified in the four tuple. Then, each one creates a Unix socket with the specified file descriptor and one socket is made to connect to the other. This procedure effectively recreates the original Unix socket connection without depending on many kernel internal details.

This use of high-level functions helps in general portability of using MobiDesk for migration. Security patches and minor version kernel revisions commonly involve modifying the internal details of the kernel while high-level primitives remain unchanged. As such services are usually made available to kernel modules through exported kernel symbol interface, the MobiDesk system is able to perform cross-kernel migration without requiring modifications to the kernel code.

The MobiDesk checkpoint-restart mechanism is also structured in such a way to perform its operations when processes are in a state that checkpointing can avoid depending on many low-level kernel details. For example, semaphores typically have two kinds of state associated with each of them: the value of the semaphore and the wait queue of processes waiting to acquire the corresponding semaphore lock. In general, both of these pieces of information have to be saved and restored to accurately reconstruct the semaphore state. Semaphore values can be easily obtained and restored through GETALL and SETALL parameters of the `semctl` system call. But saving and restoring the wait queues involves manipulating kernel internals directly. The MobiDesk mechanism avoids having to save the wait queue information by requiring that all the processes be stopped before taking the checkpoint. When a process waiting on a semaphore receives a stop signal, the kernel immediately releases the process from the wait queue and returns EINTR. This ensures that the semaphore wait queues are always empty at the time of checkpoint so that they do not have to be saved.

To provide proper support for MobiDesk virtualization when migrating across different kernels, we must ensure that that any changes in the system call interfaces are properly accounted for. As MobiDesk has a virtualization layer using system call interposition mechanism for maintaining namespace consistency, a change in the semantics for any system call intercepted by MobiDesk's session abstraction could be an issue in migrating across different kernel versions. But such changes usually do not occur as it would require that the libraries be rewritten. In other words, MobiDesk virtualization is protected from such changes in a similar way as legacy applications are protected. However, new system calls could be added from time to time. Such system calls could have implications to the pea encapsulation mechanism. For instance, across all Linux 2.4 kernels, there were two new system calls, `gettid` and `tkill` for querying the thread identifier and for sending a signal to a particularly thread in a thread group, respectively, which needed to be accounted for to properly virtualize MobiDesk across kernel versions. As these system calls take identifier arguments, they were simply intercepted and virtualized

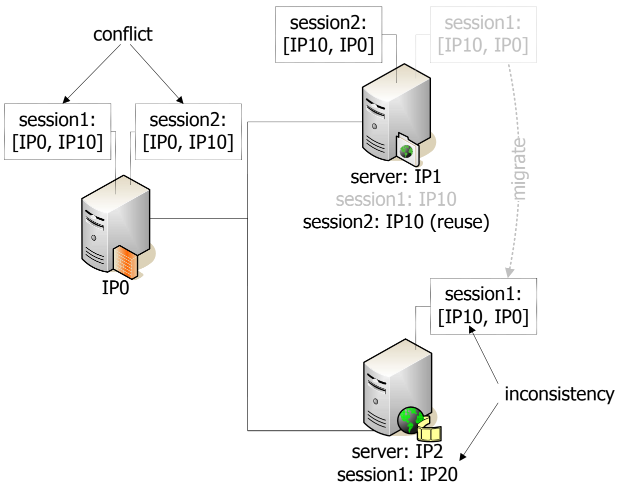

Figure 2: Key problems of connection migration

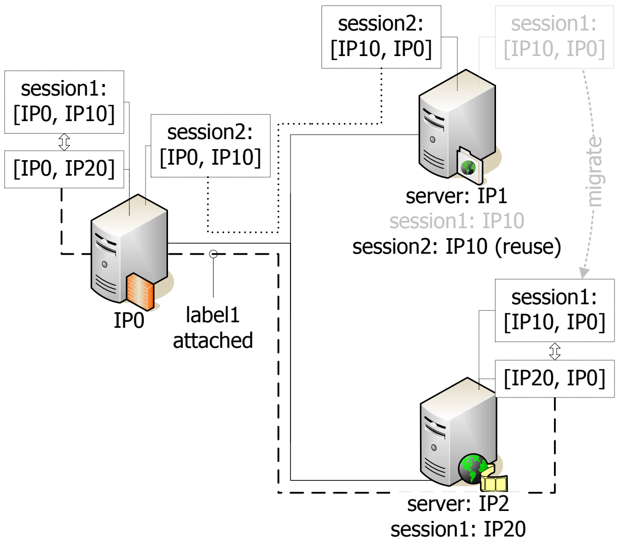

Figure 3: Mobidesk network virtualization mechanism

5 Network Virtualization

Networking support for MobiDesk sessions must address two issues:

- Multiple sessions on the same MobiDesk server may run the same service, e.g., two session may both run the apache server. However, only one of them can listen on port 80.

- Ongoing network connections of a session must be preserved when the session is migrated from one MobiDesk server to another.

When all MobiDesk servers are in the same subnet, the two issues can be addressed relatively easily using existing technologies with minor enhancements from MobiDesk. Each MobiDesk session is assigned a unique IP address from a pool maintained by a DHCP server when it's first created. From example, the MobiDesk servers may occupy IP address range 192.168.1.2 - 192.168.1.50, and the rest of 192.168.1.5 - 192.168.1.254 may be assigned to MobiDesk sessions. The IP address assigned to a session is created as an alias of the hosting server's primary IP address. Multiple aliases, each corresponding to a different MobiDesk session, can be created on a MobiDesk server. MobiDesk privatizes the aliases such that a session only sees its own alias and therefore cannot interfere traffic of other sessions on the same server.

Since each session has its own IP address, two sessions on the same server can both listen to port 80, bound to their individual private IP address. When a session is migrated from one server to another, the private IP address of the session remains unchanged; it is simply (re)created as an alias of the new hosting server's primary IP address. ARP resolves the MAC address change at the link layer and the migration is transparent to network layer and above. Ongoing network connections of the session therefore stay intact.

While it is possible to have the entire private network behind the proxy to be in a single subnet regardless of its size, it is often desirable to have separate subnets for scalability and management reasons. In this case, when a session is migrated across subnets, its private IP address can longer persist since it cannot be simply created as an alias of the new hosting server's primary IP address, which is on a different subnet. As a result, two types of problems can occur, as we illustrate in Figure 2. Note that we omit port numbers for simplicity.

We see that when the session1 with IP10 migrates from server IP1 to IP2, its transport connection [IP10, IP0] must persist but its IP address IP10 cannot since IP2 is on a different subnet; therefore creating an inconsistency. In addition, after session1 with IP10 migrates to server IP2, another session2 may reuse IP10 on server IP1 (or another server) and creates another connection [IP10, IP0]; therefore creating a conflict since the proxy will see two identical connections [IP0, IP10].

To address the inconsistency problem on the MobiDesk server, MobiDesk associates each session with two IP addresses, one is a virtual address exposed to transport and above layers and the other is a physical address seen only at network and below layers. The virtual address stays constant for the lifetime of the session while the physical address changes whenever the session migrates. The physical address is obtain from a DHCP server as in the case of a single subnet but must change when the session is migrated across subnets. There are two ways to assign the virtual address. By default, the virtual address is equal to the initial physical address when the session is created and stays unchanged thereafter. Alternatively, the virtual address can be a predefined value that is constant across all sessions. MobiDesk translates the virtual address to the current physical address (and vice versa) when the session migrates. For example, in Figure 3, after migration, session1's virtual address IP10 is unchanged while its physical address is assigned by the DHCP server to be IP20 and created as an alias on server IP2. The proxy translates [IP0, IP10] into [IP0, IP20] while the server IP2 translates [IP20, IP0] into [IP10, IP0]. Since the virtual address never changes, the migration is transparent to transport and above layers.

One potential solution to the conflict problem on the MobiDesk proxy is to require that a physical address, once assigned to a session, is never reused until the session finishes even after the session has migrated to another subnet. This however results in undesirable dependency of a session on a trail of addresses if it's migrated many times and new connections are open between each migration. MobiDesk's solution is to privatize virtual addresses, i.e., to associate virtual addresses with separate private virtual network interfaces which provide per-connection address namespace. Instead of having all connections share the same physical interface, each connection is assigned its own private virtual network interface card (VNIC). A VNIC is simply a software emulation of a NIC at the link layer that appears exactly the same as a NIC to network and above layers. As a result, two connections using the same virtual IP address due to address reuse can peacefully coexist on the same server; since they are bound to their own private VNIC.

To support per-connection address space, MobiDesk augments the traditional connection tuple with connection labels to identify the VNIC to which a connection is bound. A connection has two labels, independently and uniquely chosen by the MobiDesk proxy and the server at the time when the connection is setup. The two sides also exchange their labels at connection setup time. Before a session is migrated, the labels are not used since the tuple along is enough to identify the connections of the session. After a session is migrated, both sides will attach its peer's label learned at connection setup time for all connection between them. The labels allow the connections to be uniquely identified even when a session's previous physical address is reused. We illustrate these ideas using the example in Figure 3.

In Figure 3, when the connection [IP0, IP10] was setup for session1 while it was on server IP1, the proxy creates a VNIC for the connection and sends its label1 for the connection. to server IP1; similarly, session1 on server IP1 also sends its label for the connection to the proxy (the actual label exchange is not shown). Before session1 was migrated, its virtual address, which equals its physical address IP10, cannot be reused by other sessions ; therefore virtual address alone is enough to identify the connection [IP0, IP10] and labels are not used in the absence of migration. After session1 is migrated to server IP2, however, all packets from session1 to the proxy will have the label1 attached to them and allows the proxy to uniquely identify the connection [IP0, IP10] that belongs to session1, instead of anther session2 that reuses address IP10. For example, when session2 reuses address IP10 and creates a connection [IP10, IP0] between server IP1 and the proxy after session1 is migrated, the proxy will now have no problem letting both connections coexist peacefully; since even though they have exactly the same tuple [IP0, IP10] (at transport layer), they each are bound to a different VNIC (with the same address IP0), therefore no conflict occurs in the transport layer. Remember that when session2 creates its connection [IP0, IP10], connection labels will be exchanged between server IP1 and the proxy (not shown). But since session2 hasn't been migrated, the labels are not used and the connection is solely identified by the tuple [IP0, IP10]. In other words, packets belonging to session2's connection will not have label attached while packets belonging to session1's connection will have label1 attached. This allows the proxy to correctly identify both connections and perform virtual-physical translation as needed, e.g., session1's connection is mapped to [IP0, IP20] while no mapping is necessary for session2's connection.

6 Experimental Results

We have implemented a prototype MobiDesk system in Linux. Our prototype consists of a Linux virtual display driver that works with XFree86 and a loadable kernel module for operating system and network virtualization. Our prototype demonstrates that MobiDesk can be implemented with no changes to the Linux kernel. We present some experimental results using our Linux MobiDesk prototype to quantify its overhead and demonstrate its performance on various desktop computing applications.

Figure 4 shows the isolated network test-bed we used for our experiments. The test-bed consists of eight IBM Netfinity 4500R machines and a Micron desktop PC. The Netfinity machines each had a 933Mhz Intel Pentium-III CPU and 512MB RAM, and all of them were connected via gigabit Ethernet. The Micron desktop PC had a 450Mhz Intel Pentium-II CPU and 128MB RAM, and was used as the MobiDesk client. Four of the machines served as a MobiDesk server infrastructure consisting of one NFS file server, one proxy server running a delegate 8.9.2 general-purpose application level proxy, and two computing session servers. One

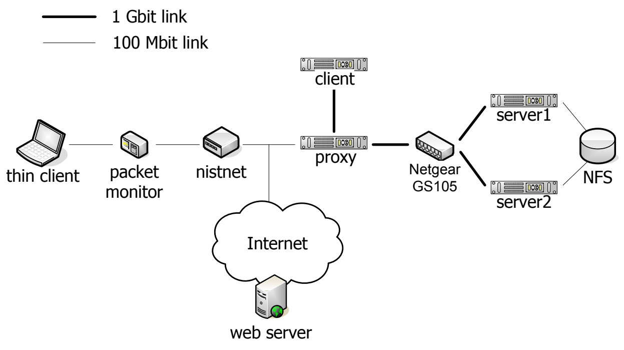

Figure 4: Experimental testbed

machine was connected on the client-side of the MobiDesk proxy and was used as a NISTNet 2.0.12 WAN emulator which could adjust the network characteristics seen by the client. Four machines were connected to the client-side of the WAN emulator, one Micron PC used as a MobiDesk client, a second used as an external web server, a third used as a packet monitor running Ethereal Network Analyzer 0.9.13 for measurement purposes, and the last used as a client for network virtualization overhead measurements. All of the machines ran Debian Linux, with the two computing session servers running Debian Stable with a Linux 2.4.5 kernel and Debian Unstable with a Linux 2.4.18 kernel, respectively. The MobiDesk client machine was installed with a dual-boot configuration and also ran Microsoft Windows XP Professional.

6.1 MobiDesk Virtualization Overhead

To measure the cost of MobiDesk's operating system virtualization, we used a range of micro benchmarks and real application workloads and measured their performance on our Linux MobiDesk prototype and a vanilla Linux system. Table 2 shows the seven micro-benchmarks and four application benchmarks we used to quantify MobiDesk's operating system virtualization overhead as well as the results for a vanilla Linux system. To obtain accurate measurements, we rebooted the system between measurements. Additionally, the system call micro-benchmarks directly used the TSC register available Pentium CPUs to record timestamps at the significant measurement events. Each timestamp's average cost was 58 ns. The files for the benchmarks were stored on the NFS server. All of these benchmarks were performed in a chrooted environment on the NFS client machine running Debian Unstable with a Linux 2.4.18 kernel. Figure 5 shows the results of running the benchmarks under both configurations, with the vanilla Linux configuration normalized to one. Since all benchmarks measure the time to run the benchmark, a small number is better for all benchmarks results.

The results in Figure 5 show that the operating system

Name	Description	Linux
getpid	average `getpid` runtime	350 ns
ioctl	average runtime for the FIONREAD ioctl	427ns
shmget-shmctl	IPC Shared memory segment holding an integer is created and removed	3361 ns
semget-semctl	IPC Semaphore variable is created and removed	1370 ns
fork-exit	process forks and waits for child which calls exit immediately	44.7 us
fork-sh	process forks and waits for child to run `/bin/sh` to run a program that prints “hello world” then exits	3.89 ms
Apache	Runs Apache under load and measures average request time	1.2 ms
Make	Linux Kernel compile with up to 10 process active at one time	224.5s
Postmark	Use Postmark Benchmark to simulate Sendmail performance	.002s
MySQL	“TPC-W like” interactions benchmark	8.33s

Table 2: Application Benchmarks

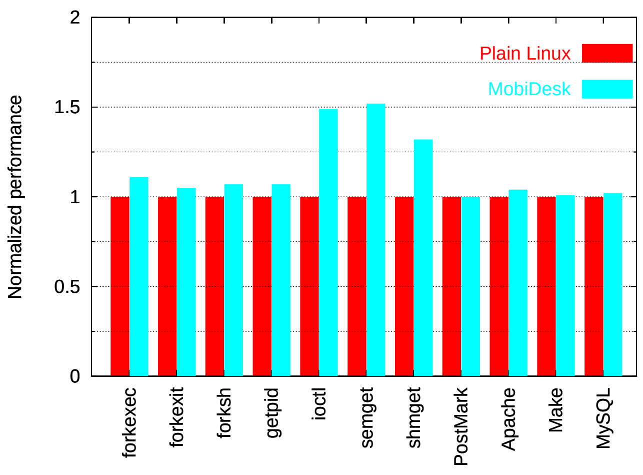

Figure 5: Operating System Virtualization Overhead

virtualization overhead is small. MobiDesk incur less than 10% overhead for most of the micro-benchmarks and less than 4% overhead for the application workloads. The overhead for the simple system call `getpid` benchmark is only 7% compared to vanilla Linux, reflecting the fact that virtualization for these kinds of system calls only requires an extra procedure call and a hash table lookup. The most expensive benchmarks for MobiDesk is `semget+semctl` which took 51% longer than vanilla Linux. The cost reflects the fact that our untuned MobiDesk prototype needs to allocate memory and do a number of namespace translations. The `ioctl` benchmark also has high overhead, because of the 12 separate assignments it does to protect the call against malicious root processes. This is large compared to the simple FIONREAD `ioctl` that just performs a simple dereference. However, since the `ioctl` is simple, we see that it only adds 200 ns of overhead over any `ioctl`. For real applications, the most overhead was only four percent which was for the Apache workload, where we used the `http_load` benchmark [25] to place a parallel fetch load on the server with 30 clients fetching at the same time. Similarly, we tested MySQL as part of a web-commerce scenario outlined by TPC-W with a bookstore servlet running on top of Tomcat with a MySQL back-end. The MobiDesk overhead for this scenario was less than 2% versus vanilla Linux.

To measure the cost of MobiDesk’s network virtualization, we used netperf 2.2pl4 to measure MobiDesk network I/O overhead versus vanilla Linux in terms of throughput, latency, CPU utilization, and connection setup. We ran the netperf client on the Netfinity client and the netperf server on one of the MobiDesk compute servers. We used the Netfinity client for these experiments instead of the MobiDesk client to so that all machines used for the network virtualization measurements were connected via gigabit Ethernet. To ensure that we were accurately measuring the performance overheads of our systems as opposed to raw network link performance, we used gigabit Ethernet for our experiments so that the network link capacity could not be saturated easily. All connections from the netperf client to the netperf server were made through the delegate proxy. We compared the performance of three different system configurations: Vanilla, MobiDesk1, and MobiDesk2. The Vanilla system is a stock Linux system without MobiDesk loaded into the kernel. The MobiDesk1 and MobiDesk2 are systems with MobiDesk loaded. On MobiDesk1, no connections are migrated and hence only connection virtualization is performed; on MobiDesk2, all connections are migrated and hence both connection virtualization and virtual-physical mapping are performed.

Figures 6 to 8 show the results for running the netperf throughput experiment, latency experiment, and connection setup experiment. CPU utilization measurements are omitted here since they show similar overhead results. The throughput experiment simply measures the throughput achieved when sending messages of varying sizes as fast as possible from the client to the server. Figure 6 shows the throughput overhead for the three systems we tested. We can see that MobiDesk1 performs very close to Vanilla, with an overhead of about 1.4Mbits/second. MobiDesk2 shows the throughput overhead due to the virtual-physical mapping, which is around 10Mbits/second.

The latency experiment measures the inverse of the transaction rate, where a transaction is the exchange of a request message of size 128 bytes and a reply message of varying sizes between the client and the server over a single connection. Figure 7 shows the latency overhead for the three systems we tested. The results bear the same characteris-

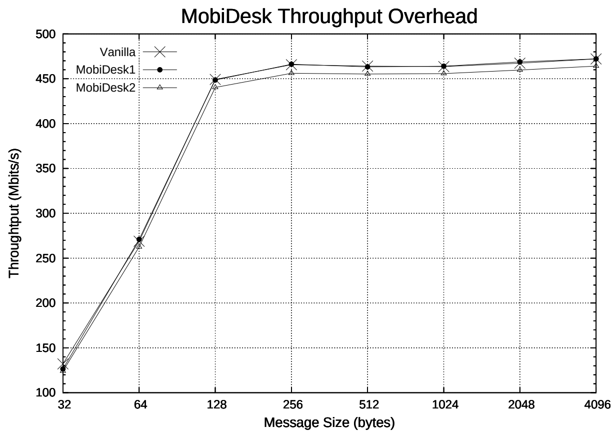

Figure 6: Virtualization Throughput Overhead

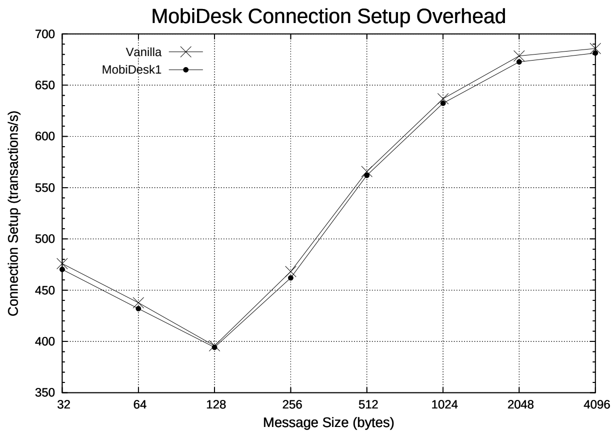

Figure 8: TCP Connection Setup Overhead

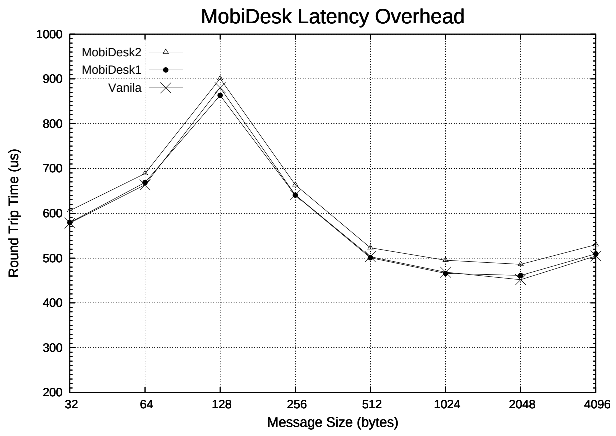

Figure 7: Virtualization Latency Overhead

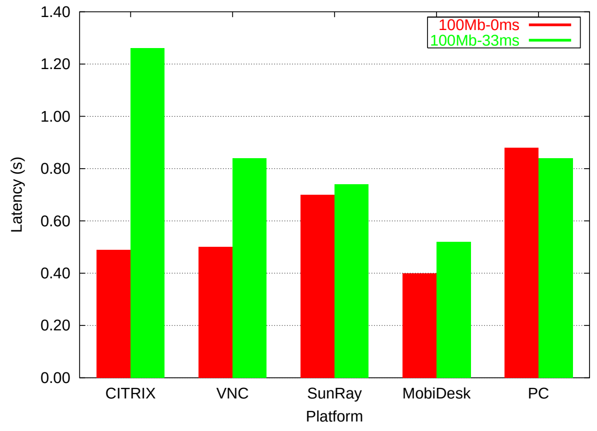

Figure 9: Web benchmark: Average per page latency

tic as that for the throughput overhead. Performance difference between Vanilla and MobiDesk1 is about 9.4 microseconds; while latency due to the virtual-physical mapping in MobiDesk2 can be observed to be around 40 microseconds. Note that there is a strange increase of latency at message size 128 bytes. The cause of this behavior is not clear to us. Interestingly, a strange drop of transaction rate is also observed in our connection setup measurement we will be presenting in the next section. We conjecture that it might have something to do with how the delegate proxy manages its buffers.

The TCP connection setup experiment is the same as the latency experiment except that a new connection is used for every request/response transaction. This experiment simulates the interaction between a client and server in which many short-lived connections are opened and closed. Figure 8 shows the TCP connection setup overhead for Vanilla and MobiDesk1. Note that since connection setup occurs before migration, there is no virtual-physical mapping overhead associated with connection setup, therefore this measurement is not applicable to MobiDesk2. From the figure we can see that the overhead is fewer than 10 transactions per second.

6.2 MobiDesk Application Performance

To evaluate MobiDesk performance on real desktop applications, we conducted experiments to measure the display performance of MobiDesk for web and multimedia applications and the migration performance of MobiDesk in moving a user's desktop computing session from one server to another. To measure display performance, we compared MobiDesk against running applications on a local PC. We also compared MobiDesk running with XFree86 4.3.0 against other popular commercial thin-client systems, including Citrix MetaFrame XP for Windows [1], VNC 3.3.7 for Linux [29], and Sun's SunRay 2.0 [37]. All of the thin-client systems except SunRay, used the Micron PC as the client and a Netfinity server as the server. SunRay used a SunRay I hardware thin client and a Solaris 9 v210 server since it does not run with the common hardware/software configuration used

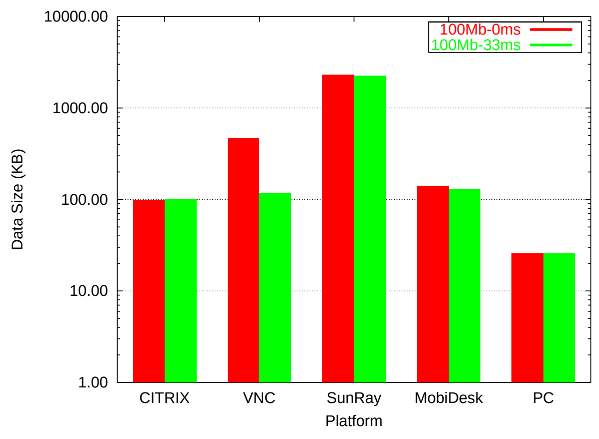

Figure 10: Web benchmark: Average per page data transfer

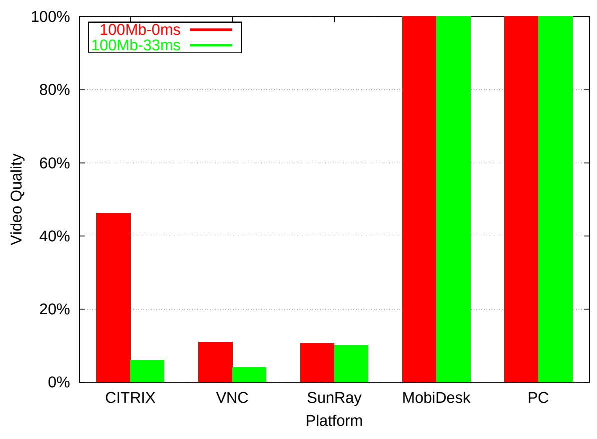

Figure 11: Video benchmark: Video Quality

by the other systems.

We evaluated display performance using two popular desktop application scenarios, web browsing and video playback. Web browsing performance was measured using a Mozilla 1.4 browser to run a benchmark based on the Web Text Page Load test from the Ziff-Davis i-Bench benchmark suite [43]. The benchmark consists of a sequence of 54 web pages containing a mix of text and graphics. The browser window was set to 1024x768 for all platforms measured. Video playback performance was measured using a video player to play a 34.75 s video clip of original size 352x240 pixels displayed at 1024x768 full screen resolution. We used the packet monitor in our testbed to measure benchmark performance on the thin-client systems using slow-motion benchmarking [20], which allows us to quantify system performance in a non-invasive manner by capturing network traffic. The primary measure of web browsing performance was the average page download latency. The primary measure of video playback performance was video quality [20], which accounts for both playback delays and frame drops that degrade playback quality. For example, 100 percent video quality means that all video frames were displayed at real-time speed. On the other hand, 50 percent video quality could mean that half the video frames were dropped when displayed at real-time speed or that the clip took twice as long to play even though all of the video frames were displayed.

For both benchmarks, we measured all systems in two representative network scenarios: LAN, with an available bandwidth of 100Mbps and no introduced latency (100Mb-0ms), and WAN, with 100 Mbps available bandwidth and 33 ms added latency (100Mb-33ms), for a 66 ms round trip time. Our WAN network environment mimics the network characteristics of high bandwidth wide area networks, such as the Internet2. For the WAN tests we increased the default TCP window size for both server and client for the platforms that used TCP. SunRay was unaffected by this since it uses UDP.

Figures 9 and 10 show the web browsing performance results in terms of the perceived latency, and average per page data transfer, respectively. Figures 11 and 12 show the video playback performance results in terms of the video quality and total data transferred, respectively. Figure 10 shows that the local PC is the most bandwidth efficient platform for web browsing, but Figure 9 shows that MobiDesk provides the smallest web page download latencies. MobiDesk outperforms the local PC in part because it leverages the faster server provided by the MobiDesk infrastructure to process web pages more quickly than the local PC running the web browser on its slower client. The results show that MobiDesk's latency-sensitive design provides faster performance than any of the other platforms. The worst web browsing platform is Citrix MetaFrame, which adopts a more high-level display approach that results in poor WAN performance because of the tight coupling required between the application running on the server and the Citrix viewer running on the client. VNC has the second worst WAN web browsing performance in part because it relies on a client pull model for sending display updates as opposed to MobiDesk's server push model, which avoids round trip latencies providing better interactive response time. In addition, as a response to the limited WAN network conditions VNC adaptively uses more efficient compression algorithms, thus reducing its data transfer, but increasing its latency, and worsening its overall web browsing performance.

Figure 12 shows that the local PC is also the most bandwidth efficient platform for video playback, but Figure 11 shows that MobiDesk provides perfect video quality in the same manner as the local PC. Figure 11 also shows that all of the other platforms deliver very poor video quality. They suffer from their inability to distinguish video data from normal display updates and apply ineffective compression algorithms on the video data, which are unable to keep up with the stream of updates being generated. In contrast, the re-

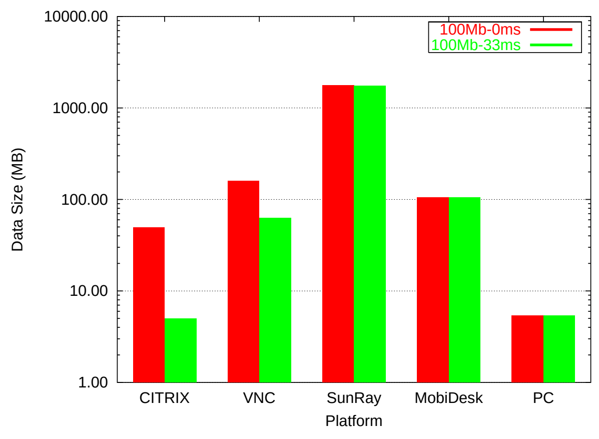

Figure 12: Video benchmark: Total data transferred at 24fps

Applications	Description
MobiDesk	Remote display server
KDE	Entire KDE 2.2.2 environment, including window manager, panel and assorted background daemon and utilities
SSH	openssh 3.4p1 client inside a KDE konsole terminal connected to a remote host
Shell	The Bash 2.05a shell running in a konsole terminal
KGhostView	A PDF viewer with a 450k 16 page PDF file loaded
Konqueror	A modern standards compliant web browser that is part of KDE
KOffice	The KDE word processor and spreadsheet programs

Table 3: Migrated KDE Desktop Computing Session

sults show that MobiDesk's ability to leverage local client video hardware in delivering video using alternative YUV formats provides substantial performance benefits over other thin-client systems. VNC provides the worst overall performance primarily because of its use of a client pull model instead of MobiDesk's server push model. In order to display each video frame, the VNC client needs to send an update request to the server. Clearly, video frames are generated faster than the rate at which the client can send requests to the server. Finally, Figure 12 shows that MobiDesk's 100% video quality does not imply high resource utilization. The total data transferred translates into a bandwidth utilization of roughly 24Mbps. While VNC and Citrix consume less bandwidth, their video quality is too low to provide useful video delivery.

To measure real application performance in terms of the cost of MobiDesk migration, we migrated a complete KDE desktop computing environment from one MobiDesk server to another. The applications running in the KDE computing session when it is migrated are described in Table 3. The KDE session had over 30 different processes running, providing the desktop applications, as well as substantial underlying window system infrastructure, including inter-application sharing, a rich desktop interface managed by a window manager with a number of applications running in a panel such as the clock. To demonstrate our MobiDesk prototype's ability to migrate a complete computing session across Linux kernels with different minor versions, we checkpointed the KDE session on the 2.4.5 kernel client machine and restarted it on the 2.4.18 kernel machine. For this experiment, the workloads were checkpointed to and restarted from local disk. The resulting checkpoint and restart times were less than a second, .85 s and .94 s, respectively. The checkpointed image was only 35 MB for a full desktop computing session, which can be easily compressed using bzip2 down to 8.8 MB. Our results show that MobiDesk can be used to provide fast migration of computing sessions among MobiDesk servers with modest checkpoint state.

7 Related Work

MobiDesk provides a utility-computing infrastructure for desktop computing. Other utility computing approaches have been proposed, primarily in the context of web services and grid computing [12, 13]. For example, IBM's Oceano project [3] proposed the use of a pool of web servers that could be reallocated to different customers based on their usage. Web services utilities focus on web applications and grid computing utilities focus on scientific applications and other applications written specifically for grids. These approaches do not support hosting general desktop computing environments, which is the focus of MobiDesk. Plan9 [23] also provides an infrastructure-based approach to desktop computing, but does not provide the same kind of mobility support with unmodified applications and operating system kernels. More generally, IBM's on-demand computing and Hewlett-Packard's utility computing initiative also demonstrate industry interest and trends toward a utility computing model.

MobiDesk display virtualization provides a display model similar to other thin-client approaches [1, 10, 29, 32, 37]. However, these approaches were not designed for WAN environments with higher network latencies. Approaches such as Citrix MetaFrame operate using higher-level display primitives that are designed more for low bandwidth environments but can result in worse WAN performance [16]. MobiDesk's virtual device driver approach is most similar to SunRay [37], but MobiDesk provides more effective mapping of display commands to protocol primitives to significantly improve performance. MobiDesk also employs additional mechanisms for latency-sensitive design including command scheduling that make its design more suitable for WAN environments.

MobiDesk operating system virtualization is similar to the Zap system [21], which supports transparent migration across systems running the same kernel version. Unlike Zap, MobiDesk is designed explicitly for supporting user computing sessions and supports transparent migration across different minor kernel versions, which is essential for providing application availability in the presence of operating system security and maintenance upgrades. Many other systems have been proposed to support process migration [28, 19, 4, 30, 11, 6, 9, 18, 26, 24, 8], but these systems do not provide general migration across independent commodity operating systems of unmodified legacy applications which use standard operating system services. TUI [34] is one of the few systems that provides support for process migration across machines running different operating systems and hardware architectures. Unlike MobiDesk, TUI has to compile applications on each platform using a special compiler and does not work with unmodified legacy applications. Virtual machine monitors (VMMs) provide an alternative virtualization approach that can be used to migrate an entire operating system environment [31]. Unlike MobiDesk, VMMs decouple processes from the underlying machine hardware, but tie them to an instance of an operating system. As a result, VMMs cannot migrate processes apart from that operating system instance and cannot continue running those processes if the operating system instance ever goes down, such as during security upgrades. In contrast, MobiDesk decouple process execution from the underlying operating system which allows it to migrate processes to another system when an operating system instance is upgraded.

MobiDesk network virtualization provides mobile communication support. Many other approaches have been developed for network mobility [7, 14, 15, 22, 27, 35, 36, 38, 40, 41, 42]. However, with the exception of [41], none of them is designed with process migration integration in mind. These approaches often also require network infrastructure support to address general mobility issues (e.g., locating a mobile host) that do not apply to the application environment of MobiDesk. Transport layer solutions such as [35, 36] can provide fine-grain connection migration without additional network infrastructure support. However, they require modifying the transport protocol (TCP in particular) itself, making them more difficult to deploy. Application layer solutions such as [27, 41, 42] can also provide fine-grain connection migration without additional network infrastructure support; neither do they require modifying the transport protocol. However, the migration is emulated by closing the old TCP connection and opening a new one. The emulation requires double buffering at the application layer to account for in-flight data that have been received by TCP but not yet delivered to the application; since these data are lost when the old connection is closed. This results in substantial network I/O overhead [41] even when the connections are not migrated. In contrast, MobiDesk employs a novel low-overhead virtualization mechanism integrated with process migration that provides network mobility without requiring network infrastructure support and transport protocol change.

8 Conclusions

We have introduced MobiDesk, an architecture for centralized hosting of desktop computing sessions. MobiDesk hosts computing sessions within virtualized and private environments by abstracting three key resources: display, operating system and network. Display virtualization allows MobiDesk to provide fast remote access to sessions across LAN and WAN environments. Operating system virtualization allows MobiDesk to migrate sessions among hosting servers to provide high availability computing in the presence of server maintenance and upgrades. Network virtualization allows MobiDesk to transparently maintain persistent connections to unmodified outside hosts, even as a session migrates from one server to another.

We have implemented and evaluated the performance of a MobiDesk prototype in Linux. Our implementation demonstrates that MobiDesk can support unmodified applications in hosted computing sessions without any changes to operating system kernels, network infrastructure, or network protocols. Our experimental results with real applications and hosted desktop computing sessions show that MobiDesk has low virtualization overhead, can migrate computing sessions with subsecond checkpoint/restart times, and provides superior display performance over other remote display systems. MobiDesk is unique in its ability to offer a complete desktop experience remotely with full-motion video support. It can even provide better performance than running a desktop session on a local PC for more resource constrained clients. Given its performance and centralized hosting model, MobiDesk provides the foundation for a utility computing infrastructure that can dramatically reduce the management complexity and costs of desktop computing.

References

[1] Citrix ICA Technology Brief. Technical White Paper, Boca Research, Boca Raton, FL, 1999.

[2] The 100x100 Project. `http://100x100network.org/`.

[3] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, and M. Kalantar. Oceano: SLA based Management of a Computing Utility. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management*, May 2001.

[4] Y. Artsy, Y. Chang, and R. Finkel. Interprocess communication in charlotte. *IEEE Software*, pages 22–28, Jan 1987.

[5] N. Bansal and M. Harchol-Balter. Analysis of SRPT scheduling: investigating unfairness. In *SIGMETRICS/Performance*, pages 279–290, 2001.

[6] A. Barak and R. Wheeler. MOSIX: An Integrated Multiprocessor UNIX. In *Proceedings of the USENIX Winter 1989 Technical Conference*, pages 101–112, San Diego, CA, Feb. 1989.

[7] P. Bhagwat and C. Perkins. A Mobile Networking System based on Internet Protocol (IP). In *Proceedings of USENIX Symposium on Mobile and Location Independent Computing*, pages 69–82, Cambridge, MA, Aug. 1993.

[8] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A migration transparent version of PVM. *Computing Systems*, 8(2):171–216, 1995.

[9] D. Cheriton. The V distributed system. *Communications of the ACM*, 31(3):314–333, Mar 1988.

[10] B. Cumberland, K. Schauser, and M. Munke. *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Microsoft Press, Redmond, WA, Aug. 1999.

[11] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementatio. *Software - Practice and Experience*, 21(8):757–785, Aug. 1991.

[12] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.

[13] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz. Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. In *Cluster Computing*, volume 5, 2002.

[14] J. Ioannidis, D. Duchamp, and G. Q. Maguire. IP-based Protocols for Mobile Internetworking. In *Proceedings of ACM SIGCOMM*, pages 235–245, 1991.

[15] D. B. Johnson and C. Perkins. Mobility support in ipv6. *draft-ietf-mobileip-ipv6-16.txt, IETF*, Mar. 2002.

[16] A. Lai and J. Nieh. Limits of Wide-Area Thin-Client Computing. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2002)*, pages 228–239, Marina del Rey, CA, June 2002.

[17] D. Legard. Korea to build 100Mbps Internet system. *InforWorld*, Nov.18 2003. `http://www.infoworld.com/article/03/11/18/HNkorea_1.html`.

[18] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical Report 1346, University of Wisconsin Madison Computer Sciences, Apr. 1997.

[19] S. J. Mullender, G. v. Rossum, A. S. Tanenbaum, R. v. Renesse, and H. v. Staveren. Amoeba a distributed operating system for the 1990s. *IEEE Computer*, 23(5):44–53, May 1990.

[20] J. Nieh, S. J. Yang, and N. Novik. Measuring Thin-Client Performance Using Slow-Motion Benchmarking. *ACM Transactions on Computer Systems (TOCS)*, 21(1):87–115, Feb. 2003.

[21] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec. 2002.

[22] C. Perkins. IP Mobility Support for IPv4, revised. *draft-ietf-mobileip-rfc2002-bis-08.txt, Internet Draft*, Sept. 2001.

[23] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. Technical White Paper, Bell Laboratories, Murray Hill, New Jersey, 1995.

[24] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of Usenix Winter 1995 Technical Conference*, pages 213–223, New Orleans, LA, Jan 1995.

[25] J. Poskanzer. `http://www.acme.com/software/http_load/`.

[26] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *2nd Workshop on Job Scheduling Strategies for Parallel Processing (In Conjunction with IPPS '96)*, Honolulu, Hawaii, Apr. 1996.

[27] X. Qu, J. X. Yu, and R. P. Brent. A Mobile TCP Socket. In *International Conference on Software Engineering (SE '97)*, San Francisco, CA, Nov. 1997.

[28] R. Rashid and G. Robertson. Accent: A communication oriented network operating system kernel. In *Proceedings of the 8th Symposium on Operating System Principles*, pages 64–75, Dec 1984.

[29] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, Jan-Feb 1998.

[30] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the Chorus distributed operating system. In *Workshop on Micro-Kernels and Other Kernel Architectures*, pages 39–70, Seattle WA (USA), 1992.

[31] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[32] R. W. Scheifler and J. Gettys. *X Window System*. Digital Press, third edition, 1992.

[33] B. Schneier. *Applied Cryptography*. John Wiley and Sons, second edition, 1996.

[34] P. Smith and N. C. Hutchinson. Heterogeneous process migration: The Tui system. *Software – Practice and Experience*, 28(6):611–639, 1998.

[35] A. C. Snoeren and H. Balakrishnan. An End-to-End Approach to Host Mobility. In *Proceedings of 6th International Conference on Mobile Computing and Networking (MobiCom'00)*, Boston, MA, Aug. 2000.

[36] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory TCP: Highly available internet services using connection migration. In *Proceedings of ICDCS*, pages 17–26, 2002.

[37] Sun Ray Integrated Solutions. `http://www.sun.com/products/sunray1/`.

[38] F. Teraoka, Y. Yokote, and M. Tokoro. A Network Architecture Providing Host Migration Transparency. In *Proceedings of ACM SIGCOMM*, Sept. 1991.

[39] VMware, Inc. `http://www.vmware.com`.

[40] P. Yalagandula, A. Garg, M. Dahlin, L. Alvisi, and H. Vin. Transparent Mobility with Minimal Infrastructure. In *Technical Report 01-30, University of Texas at Austin*, June 2001.

[41] V. C. Zandy and B. P. Miller. Reliable Network Connections. In *Proceedings of 8th ACM International Conference on Mobile Computing and Networking (Mobicom '02)*, Atlanta, GA, Sept. 2002.

[42] Y. Zhang and S. Dao. A Persistent Connection Model for Mobile and Distributed Systems. In *4th International Conference on Computer Communications and Networks (ICCCN)*, Las Vegas, NV, Sept. 1995.

[43] i-Bench version 1.5, Ziff-Davis, Inc. `http://i-bench.zdnet.com`.