Leveraging Smart Phones to Reduce Mobility Footprints

Stephen Smaldone[†] Benjamin Gilbert[•] Nilton Bila[⋆]
Liviu Iftode[†] Eyal de Lara[⋆] Mahadev Satyanarayanan[•]

[†]Rutgers University [•]Carnegie Mellon University [⋆]University of Toronto

smaldone@cs.rutgers.edu bgilbert@cs.cmu.edu nilton@cs.toronto.edu
iftode@cs.rutgers.edu delara@cs.toronto.edu satya@cs.cmu.edu

ABSTRACT

Mobility footprint refers to the size, weight, and energy demand of the hardware that must be carried by a mobile user to be effective at any time and place. The ideal of a zero mobility footprint is achievable by encapsulating personal computing state in a virtual machine (VM) and delivering it over the Internet to a locally-obtained computer close to the user. In locations with poor Internet connectivity, the demands placed on WAN bandwidth can result in unacceptable user experience. We show how this challenge can be overcome by using nascent smart phone technology as a trusted personal assistant called *Horatio* that serves as *a self-cleaning portable cache for VM state.* Since most users already carry cell phones for voice calls and texting, Horatio does not increase the size or weight aspects of a user's mobility footprint — there is only a small increase in the energy aspect. We have built an experimental prototype of Horatio, and measurements confirm its ability to improve user experience even with current smart phone limitations.

Categories and Subject Descriptors

D.4.7 [**Organization and Design**]: Distributed systems

General Terms

Design, Experimentation, Performance, Reliability

Keywords

Horatio, self-cleaning, portable, cache, smart phone, ISR, OpenISR, Internet Suspend/Resume, mobile computing, virtual machine, content addressable storage, CAS

1. INTRODUCTION

The term "mobile computing" spans many different approaches to accessing one's personal computing (PC) state on the go. Although "carry all the hardware you might need" has been the dominant approach since the early 1990s, new models of computing have emerged that allow users to carry less by relying on hardware resources available at sites along their path of travel. Carrying less hardware reduces a user's *mobility footprint:* that is, the size, weight, and energy demand of what must be carried to be effective on the go. The holy grail of mobile computing is to combine a tiny mobility footprint with complete confidence in the performance, reliability and safety of precisely re-creating one's PC state at any time and place.

A "carry nothing and live off the land" approach is offered by both thin-client remote access products, such as *GoToMyPC* [1], and the *Internet Suspend/Resume*® *(ISR)* approach [9, 13]. In thin client remote access, a server executes application and operating system code while the client only performs screen updates and user interactions. In ISR, PC state within a hardware virtual machine (VM) is delivered from a server over the Internet for execution on local hardware. The ISR approach is more network resilient: user experience is good even with high network latency and jitter, and disconnected operation is possible. These are important considerations for mobile computing at planetary scale.

ISR's zero mobility footprint comes at the price of large VM state transfers. Even if this state is fetched lazily from a server, it is likely to involve many tens or hundreds of MB at startup, with further transfers during execution. This results in significant startup delay ("resume latency"), and slower execution ("slowdown"). When a user finishes work, modified VM state has to be transferred back to the server. In trusted environments such as home or office, the user can depart without waiting for this transfer to complete. But in public environments such as an Internet cafe, the prudent user suffers a final delay ("suspend latency"). Resume and suspend latencies tend to be more noticeable than slowdown.

In this paper, we show how the storage and Internet connectivity of smart phones can be used to alleviate these ISR limitations. Our design treats smart phones as trusted personal assistants that serve as *self-cleaning portable caches for VM state.* We call such an assistant *Horatio,* alluding to Hamlet's trusted ally in Shakespeare's play. An ISR user can suspend his VM state to Horatio rather than directly to the server; similarly, he can resume from Horatio. Even when Internet connectivity is poor, the physical proximity of Horatio to the client ensures good connectivity between them: for example, a USB 2.0 cable or one of the emerging wireless technologies such as Ultra-Wideband (UWB). To reduce device vulnerability, Horatio opportunistically uses cellular, WiFi or other networking technology to asynchronously propagate modified VM state to ISR servers while users are in transit. This self-cleaning aspect of Horatio distinguishes it from approaches such as *SoulPad* [5] that rely solely on passive USB storage, and are hence vulnerable to device loss or damage. Horatio does not increase the size or weight aspects of ISR's mobility footprint because most users already carry cell phones for voice calls and texting. Only the energy aspect increases slightly.

We have implemented a prototype of Horatio that runs on two currently-available smart phones: the Symbian-based Nokia N95 and the Linux-based Openmoko Neo FreeRunner. Measurements with this prototype confirm that current smart phone technology is already adequate for improving user experience in ISR settings. At the same time, our measurements reveal significant inefficiencies and suggest improvements in the protocol stacks and software environments of current smart phones.

This paper makes five contributions:

- It introduces mobility footprint as a fundamental attribute of usability, and explores the merits of alternative approaches to improving this attribute in mobile computing systems.
- It extends the well-known two-tier client-server architecture to include a smart phone as an intermediate third tier that operates as a mobile self-cleaning cache. This extension improves user-perceived performance while preserving the classic two-tier architecture's strengths in security, performance and manageability.
- It describes the design and prototype implementation of a system with small mobility footprint that is based on this extended client-server architecture.
- It demonstrates the usability benefits of this architectural extension through extensive experiments on the prototype implementation.
- It identifies specific improvements to current smart phones to better serve as mobile self-cleaning caches.

The remainder of the paper is organized as follows. Sections 2 and 3 present relevant background, provide motivating examples, and position Horatio within a taxonomy of design choices. Section 4 presents the design and prototype implementation of Horatio. Section 5 presents an experimental evaluation of the prototype. Section 6 discusses some open issues and future research directions. Finally, Section 7 concludes the paper.

2. BACKGROUND AND MOTIVATION

2.1 ISR Overview

As its name suggests, ISR emulates the suspend/resume capability of laptop hardware. This is a well-understood metaphor today, and one that applications and operating systems already support. The difference is, of course, that ISR allows one to suspend on one machine and seamlessly resume on another. ISR achieves this functionality by layering a VM-encapsulated computing environment, called a *parcel,* on distributed storage based on the client-server model. A parcel corresponds to the entirety of PC state, including CPU, memory, disk, and I/O devices. It thus includes a guest OS, installed applications, and local files. Further discussion of these concepts and a summary of ISR evolution can be found in a recent paper [13]. This work was based on release 0.9.4 of the *OpenISR*® platform, whose source code can be downloaded from `http://isr.cmu.edu`.

2.2 Motivating Examples

To illustrate how Horatio can improve user experience, we provide two hypothetical ISR scenarios that capture the essence of real-world usage.

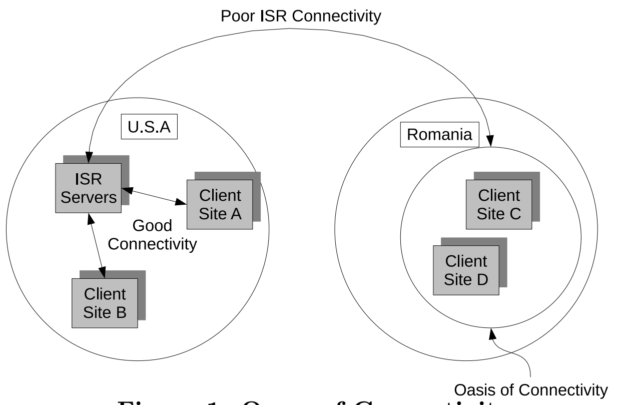


Figure 1: Oases of Connectivity

2.2.1 *A Day in a Busy Professional's Life*

Jill is a young professional. On a typical morning, she does homework on a desktop for her part-time MBA course and then checks her e-mail before leaving for work. During her commute, her parcel is suspended to an ISR server. At work, she resumes her desktop session and has a productive morning. Over lunch, she finishes her homework and then downloads and watches an episode of her favorite TV sitcom. After work, Jill meets friends at a coffee shop to view and edit vacation pictures on a laptop loaned by the coffee shop. Before heading to the gym, she updates her iPod for the workout with a podcast delivered earlier to her parcel.

In this scenario, ISR readily supports Jill's well-connected desktop sessions at home and office. However, resume and suspend times at the coffee shop are unacceptable because of poor Internet connectivity. Since the loaner laptop has to be returned to the coffee shop, Jill is forced to wait for the full duration of the suspend latency.

Horatio could greatly improve Jill's user experience at the coffee shop. When leaving work, Jill suspends her desktop session to her smart phone. At the coffee shop, she quickly resumes from the phone. Before leaving the coffee shop, she quickly suspends to her phone, returns the loaner laptop and heads off for the gym. During her walk, Horatio uses the phone's 3G connectivity to send parcel changes to the ISR server; it completes this task during her workout by using WiFi connectivity in the gym. Together, ISR and Horatio provide Jill with the illusion of seamless and ubiquitous access to her PC state without her carrying anything more than her smart phone.

2.2.2 *A Week in a Global Traveler's Life*

Jack is a marketing consultant with projects that span the U.S. and Europe. An upcoming business trip requires him to make multiple stops within the U.S. and then multiple stops within Romania. As Figure 1 illustrates, Jack will experience two *oases of connectivity* (entirely within the U.S., and entirely within Romania) within which Internet connectivity is excellent. However, connectivity is poor between the U.S. and Romania. When Jack travels between client sites A and B within the U.S., he can resume his ISR session directly from the server; when he is ready to leave a site, he can suspend his ISR session back to the server. However, when Jack travels between sites in Romania (C and D in Figure 1), his ISR user experience is unacceptable. Because of poor connectivity to his ISR server in the U.S., he experiences long resume latency, substantial slowdown and long suspend latency. Thus, ISR becomes virtually unusable.

	Carry PC hardware	Thin client remote access	Software virtualization	VM on USB device	VM delivered over Internet	Horatio
Examples of approach	any laptop, notebook, or UMPC	GoToMyPC, VNC	MojoPac, U3, Ceedo	SoulPad	ISR	-
Mobility footprint	large	zero	very small	very small	zero	small
Accuracy of re-creation	perfect	perfect	variable	high	high	high
Network resiliency	perfect	low	perfect	perfect	high	very high
Network demand	zero	low	zero	zero	very high	high
Physical vulnerability	high	zero	high	high	very low	low

Table 1: Strengths and Weaknesses of Mobile Computing Alternatives

Horatio offers a powerful solution to this problem. Before leaving the U.S., Jack can suspend his session to Horatio. In Romania, he can resume directly from Horatio. As Jack moves from Site C to Site D, for example, he can quickly suspend and resume to and from Horatio, with performance similar to what he experiences while in the U.S. Additionally, as he travels within Romania, Horatio can opportunistically take advantage of transient good connectivity to the U.S. to incrementally propagate modified state to his ISR server. Finally, when Jack returns to the U.S., Horatio can complete any remaining state transfer and synchronization steps.

3. TAXONOMY OF APPROACHES

Table 1 summarizes how well alternative approaches to mobile computing approximate the unattainable ideal of a zero mobility footprint combined with perfect reliability, safety and performance. Although both ISR and thin client remote access offer a zero mobility footprint, ISR has much lower sensitivity to WAN latency and jitter. This arises from the very different ways in which ISR and thin clients use the network. With ISR, synchronous network access is needed only to service cache misses. Once a part of VM state is cached, further execution only involves local accesses to it. After the working set of VM state is built up, an ISR user's interactive experience on applications and files within the guest OS is fully insulated from network degradation until the end of her session. Only references to remote files or to the Web continue to be network-dependent. In contrast, thin clients are sensitive to network degradation for the entire duration of a user session. Tolia *et al.* [17] show that the adequacy of thin-client computing is highly variable, and depends on both the application and the available network quality. It is latency, not bandwidth, that is the greater challenge. Tightly coupled tasks such as graphics editing suffer more than loosely coupled tasks such as web browsing. ISR makes the tradeoff of reducing sensitivity to network latency and jitter at the cost of increased bandwidth demand: shipping VM state involves far more data than the keystrokes, mouse movements, and screen updates of thin client remote access.

ISR's tradeoff (increasing bandwidth demand for reduced sensitivity to WAN latency and jitter) aligns well with the current trajectory of Internet evolution. It is very unlikely that the fundamental considerations leading to this tradeoff will change in the foreseeable future. The prime targets of networking improvements today are bandwidth, security, energy efficiency, and manageability. Often, the techniques used for these improvements hurt latency. For example, firewalls and overlay networks both achieve their goals by increasing the software path length traversed by packets. In wireless networks, a common energy-saving technique is to turn on the mobile device's transceiver only for short periods of time to receive and acknowledge packets that have been buffered at a base station. This increases average end-to-end latency for packets, and also increases jitter. In contrast, bandwidth may be hardly affected by these techniques because it is an aggregate rather than instantaneous measure. WAN bandwidth is likely to improve over time, but comparable improvements in WAN latency and jitter are unlikely.

Total network independence (i.e., perfect resilience and zero demand) combined with a very small mobility footprint can be achieved by transporting PC state on a USB storage device. This corresponds to the columns labeled "Software virtualization" and "VM on USB device" in Table 1. For example, in *SoulPad* [5] execution state is encapsulated in a VM and transported on a USB storage device between different machines. *MojoPac* [2] is a software virtualization product that customizes applications for installation on to a USB storage device, which can then be transported to any other compatible machine for re-creation of the original application environment. Accuracy of re-creation is the property that defines the level to which the user experience is similar across machines.

Physical vulnerability is a concern with all approaches except thin clients. With ISR, it only involves loss of work since the last suspend. With the other approaches, loss or destruction of the USB storage device leads to loss of PC state. A careful regimen of backups can help, but many users are not sufficiently disciplined for this to be a satisfactory solution. From the viewpoint of usability, restoration from backup is typically an administrative step that involves different software, different system context and a different set of commands and actions from normal use. In contrast, recovery in ISR is integrated with normal usage context: one merely asks to resume an earlier version of a parcel. The lack of integration also requires a backup system to be run periodically to scan for state changes, even when there are no changes. This introduces a trade-off between energy usage and time between backups on a smart phone.

Horatio improves upon ISR along the dimensions of network resiliency and network demand. It achieves this at a slight increase in the mobility footprint. There is also a slight increase in physical vulnerability because dirty VM state may reside on a Horatio device while awaiting transfer to a server. However, the self-cleaning aspect of Horatio bounds the duration of physical vulnerability. If dirty state on Horatio is lost before self-cleaning completes, only the work done since the most recent resume is lost. Compared to Horatio, the "Software Virtualization" and "VM on USB device" approaches have greater physical vulnerability, since entire computing state resides on a local USB storage device. If the device is lost or damaged, the only recourse is to restore from backup. This is more onerous than simply executing "`isr resume <previous-parcel-version>`."

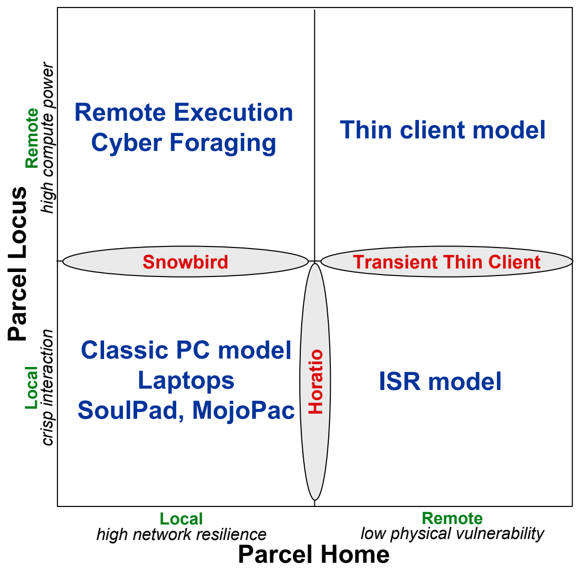


Figure 2: Separating Parcel Storage and Execution

Figure 2 illustrates a different way of looking at the design space around Horatio. For ease of exposition, we use the term "parcel" in this figure to mean "user's computing environment" even in non-ISR models. We also use the terms "resume" and "suspend" to mean execution startup and shutdown. The design space has two dimensions. The *parcel home* dimension corresponds to the storage site of a parcel when it is not in active use. The *parcel locus* dimension corresponds to the current execution site of a parcel. Moving clockwise from bottom left, the quadrants of Figure 2 map to different models of mobile computing:

- LOCAL/LOCAL: *local hardware executes a local parcel.* The classic unvirtualized PC falls into this quadrant. A laptop has a smaller mobility footprint, but is otherwise identical. Even smaller mobility footprints are exhibited by SoulPad, MojoPac and other solutions based on a small USB device. This quadrant exhibits fast resume and suspend combined with crisp interactive response that is immune to network quality and reliability. However, the parcel is vulnerable to damage or loss of its storage device.
- LOCAL/REMOTE: *remote hardware executes a local parcel.* When the remote resources are dynamically discovered in mobile computing, this corresponds to "cyber foraging" [4]. A small mobility footprint is achieved because the mobile device only needs to store the parcel rather than to execute it. Device vulnerability is similar to the local/local case.
- REMOTE/REMOTE: *parcel is both stored and executed remotely.* The client merely serves as a portal to the execution. It thus corresponds to thin client remote access, and provides fast resume and suspend with a zero mobility footprint and zero device vulnerability. On the negative side, it has low network resiliency.
- REMOTE/LOCAL: *a remote parcel is fetched for local execution.* This is essentially the ISR model. Resume and suspend are potentially slower than the local/local case, but the mobility footprint is zero and device vulnerability is low.

Figure 2 also identifies three hybrid models. "Snowbird" is a VM-based system [10] that dynamically morphs between local and remote execution. "Transient thin client" is a planned extension of the OpenISR platform that will allow a user to quickly resume a parcel via thin client remote access [13]; execution morphs into local execution once the working set of parcel state has been transferred to the client in the background. "Horatio" corresponds to the case where a smart phone serves as a temporary parcel home.

4. DESIGN AND IMPLEMENTATION

The enabling technology for Horatio is the *smart phone,* whose primary function includes basic mobile phone capabilities such as voice calls and texting. In addition, these smart phones may be viewed as "always-on" computing and storage devices with multiple network modalities (such as 3G, WiFi and Bluetooth) for Internet connectivity. Since these devices are still in the early stages of evolution, their capabilities are likely to improve significantly over time. Our goal in this research is to both validate the Horatio concept on currently-available smart phones, and to identify specific directions of improvement for future smart phones to serve as Horatio devices.

Horatio's goal is to serve as a performance accelerator for ISR, and to thus improve user experience in situations where Internet connectivity is poor or non-existent. In other words, Horatio should reduce resume latency, slowdown, and suspend latency in scenarios such as those in Section 2.2. It should also enable use of a machine with no Internet connectivity as an ISR client. It should achieve these benefits without significantly increasing the mobility footprint (i.e., energy usage) of a smart phone.

To meet these requirements, our design and implementation are based on a few key principles. We list these below, and discuss them in Sections 4.2 to 4.6:

- *Expose opportunities for parallelism, asynchrony and speculation in data transfers by separating control from data (Sections 4.2, 4.6.1, and 4.6.3).*
- *Recognize that trust flows upstream, while performance flows downstream (Section 4.3).*
- *Use client and server resources rather than Horatio resources whenever possible (Section 4.4).*
- *Keep Horatio clean (Section 4.5).*

4.1 Design Assumptions

We have made the following four assumptions in designing Horatio. First, Horatio operates within an ISR-enabled environment and uses the remote parcel home model of Figure 2. Consistent with this model, a Horatio user requires temporary use of clients wherever needed, and establishes or implicitly assumes trust in those clients prior to use [12, 14].

Second, we assume a Horatio user carries a smart phone that supports wireless networking (3G and WiFi), and has a USB-mountable disk large enough to store a user's parcel data. This assumption is realistic since modern smart phones already support multiple wireless networking options, and many come with 8GB or more of disk (some are even upgradeable to over 16GB with microSD disks). For both 3G and WiFi, we assume a fixed-fee cost model rather than a volume-sensitive model for wireless data transfers.

Our third assumption is that users place extended trust in their smart phones, but only transient trust in ISR clients that are borrowed. Between resume and suspend, the user trusts the ISR client that is executing her parcel. Once she walks away after suspend, she no longer counts on that client

State Name	Type	Typical Size	Description
Memory Image	data	200 MB	Encrypted and compressed memory image and registers.
Disk Image	data	3.5 GB	Individually encrypted and compressed chunks of virtual disk.
Keyring	control	5.5 MB	Encryption keys and cryptographic hashes of virtual disk chunks. Encrypted with a key stored in the configuration file.
Configuration File	control	500 bytes	Operational parameters of a parcel and encryption key used to encrypt the keyring and memory image.
Ownership Nonce	control	10 bytes	A unique identifier generated when a parcel is checked out from an ISR server.

Table 2: Data and Control State of a Parcel

to remain uncompromised or to propagate dirty state to the server. In contrast, she has complete confidence that Horatio will propagate dirty state to the server even if it takes many hours or days. Although attacks on smart phones are growing, this problem needs to be solved even if the smart phone is not used as a Horatio device.

Fourth, we assume strong connectivity between the ISR client and Horatio, which is reasonable given the close proximity of Horatio to the ISR client. For the ISR client-server and Horatio-ISR server links, we allow for the broadest range of connectivity, including total disconnection.

4.2 Data and Control Separation

Achieving Horatio's twin goals of improving ISR user experience and efficient self-cleaning is complicated by several factors. These include the large size of VM state, the wide range of network connectivity that has to be tolerated (ranging from gigabit LANs down to kilobit wireless links and even total disconnection), and the unpredictability in the availability and durability of network connections. When WiFi coverage is available, it is preferable to 3G for self-cleaning both from the viewpoint of performance and energy usage. When WiFi is unavailable, 3G may be the only choice. In addition, a recently-used ISR client may sometimes assist in propagating dirty state after suspend, even though our trust model does not require it to provide this assistance.

In other words, a good Horatio design has to work robustly and efficiently even in the face of considerable uncertainty in connectivity and client participation. We address these requirements by enabling uncoordinated, asynchronous data transfers to take place in parallel to the server from one or more recently-used clients as well as Horatio. Our design follows the principle of *opportunism* advocated by Tolia *et al.* in their work on DOT [16]. As their work shows, the key to achieving efficiency while preserving correctness with minimal coordination across multiple data transfer channels is to cleanly separate bulk data from its meta-data.

As Table 2 illustrates, Horatio views a parcel as consisting of two very distinct components: *data state* and *control state*. The data state consists of an encrypted VM memory image and individually-encrypted 128KB chunks of the virtual disk. The control state, which is the knowledge necessary to decrypt, validate and use the data state, consists of an encrypted keyring, a configuration file, and an *ownership nonce*, which is described below. Clean separation of control and data simplifies the use of replication and parallelism, while enabling the speculative transmission of modified data to be performed through the use of eager state propagation. Horatio can be quite cavalier in its use of these techniques for data state, provided it handles control state very carefully. Figure 3 illustrates client-Horatio-server interactions, which are explained in more detail in Sections 4.2.1 to 4.2.3.

4.2.1 Parcel Ownership and Handoff

The absence of state sharing across VMs means that a single exclusive lock per parcel is acceptable. In Horatio, the ownership nonce represents this lock. When a parcel is checked out, the ISR server generates a new ownership nonce. This nonce has to be provided when the parcel is checked in. Possession of the nonce is proof that the entity performing the checkin is acting on behalf of the user. This is a reasonable assumption since all network connections (client-server, client-Horatio, and Horatio-server) are encrypted and authenticated. Exactly one site (client, server, or Horatio) can own a particular parcel at any point in time. Only the parcel owner can decrypt and validate the data state, but any site can cache parts of the data state or transmit parts of it in any way it chooses. As a result, most of a parcel's data state can be transferred opportunistically by the site most able to do so. Regardless of how it reached its destination, such data can be safely used after validation.

Ownership handoff occurs in three steps. First, the source site confirms that all required data has arrived at the destination site. Second, the keyring and configuration file are transferred. Finally, the ownership nonce is transferred using a two-phase commit protocol. If the source site is a client or Horatio, it deletes its copy of the control state, rendering it unable to decode any data state that is still cached.

4.2.2 Reducing Suspend Latency

When client-server bandwidth is poor, the user can save time by performing a checkin of his parcel to his Horatio device rather than directly to the server. This can occur over any available connectivity such as USB, Ethernet, WiFi or Bluetooth. Because the bandwidth between the client and Horatio is likely to be much better than the bandwidth between the client and server, checking in to Horatio can significantly decrease suspend latency.

The client first transmits modified data state to Horatio, then transfers parcel ownership. After this point, the client may still have modified data state in its cache and may continue transmitting this dirty state directly to the server after the user departs. This is not required for correctness, but can reduce Horatio's mobility footprint by reducing the volume of wireless data it needs to transmit.

4.2.3 Reducing Resume Latency

Horatio can reduce resume latency by serving as a *lookaside cache* [15] for data state. If the parcel is currently owned by the server, the control state is fetched from there. Otherwise, it is fetched from Horatio. Using the control state, the cryptographic hashes of the memory image and virtual disk chunks can be obtained. These are used to demand-fetch parts of data state from Horatio, if possible. Sometimes, Horatio may not possess parts of the data state (typically

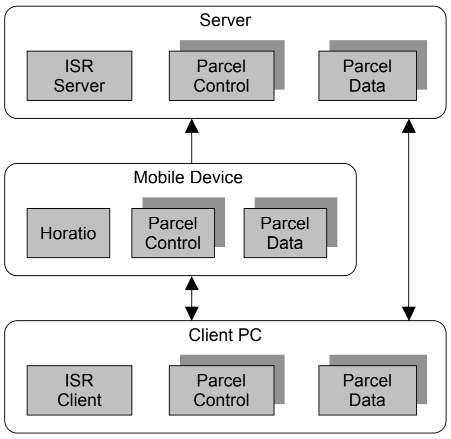


Figure 3: Control and Data Transfer in Horatio

those parts that are unmodified relative to the parcel version stored on the server). In that case, the lookaside reference fails and the client fetches that data directly from the server.

Horatio can be used to resume a parcel on a client that is disconnected from the Internet. For this to be successful, the user must have performed two steps in anticipation of this possibility. First, he should have transferred ownership of the parcel to Horatio. Second, he should have *hoarded* [8] all data state on Horatio.

4.3 Trust versus Performance Flow

The separation of data from control, together with the extensive use of parallelism, asynchrony and speculation in data transfers, results in a large and complex distributed parcel state space that spans an ISR server, multiple clients and a Horatio device. Reasoning about correctness in this state space (for example, "Can site A transfer ownership of parcel X to site B right now?") requires a simple set of rules that is consistent with user intuition. Our rules are based on a trust-performance hierarchy that arises from the introduction of a trusted portable device into the client-server model.

At the apex of trust is an ISR server. By definition, a server is the ultimate authority for all the parcels that it owns. It contains the definitive, complete and most recent states of those parcels, except for some periods of time when a parcel has been checked out on a client or suspended to Horatio. Although the server may be temporarily inaccessible due to poor Internet connectivity, it is assumed to be completely trustworthy and reliable.

In the eyes of its owner, a Horatio device is completely trustworthy. It is thus ideally placed to offer ISR-related services to its owner even when the server is inaccessible or poorly-connected. For example, Horatio can buffer updates for asynchronous propagation to a server. This resembles the role envisioned for *waystations* by Kim *et al.* in their work on Fluid Replication [7].

Lowest in the trust hierarchy is an ISR client at which a user has checked out a parcel. The decision to trust the client is made by the user based on familiarity (such as a machine at work or home), reputation (such as use of a machine at an Internet cafe that is known to provide malware-free machines), or an explicit trust establishment procedure [6, 14].

Counter to this trust hierarchy, which flows upstream (from client, through Horatio, to server) is a performance hierarchy that flows downstream. State that is already cached at the

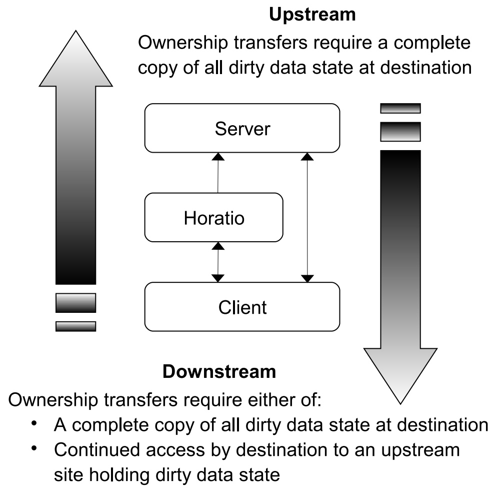

Upstream
Ownership transfers require a complete copy of all dirty data state at destination
Server
Horatio
Client
Downstream
Ownership transfers require either of:
• A complete copy of all dirty data state at destination
• Continued access by destination to an upstream site holding dirty data state

Figure 4: Rules for Ownership Transfer

client has the lowest access cost and hence highest performance. State on Horatio is slower to access, but often much faster than accessing state on a distant server. Finally, state on the server is most complete, but slowest to access.

These considerations of trust and performance lead to a set of simple rules pertaining to ownership transfer. We distinguish between "upstream" and "downstream" ownership transfers as shown in Figure 4. In the case of downstream transfers, while the downstream device must have access to all parcel state in order to resume the parcel, the state can be fetched on demand. Therefore, the downstream site must either have ongoing access to an upstream site holding all of the data state, or must hoard all of that data state in advance. Dirty state can be held by an upstream site under the same rules. When transferring ownership upstream, on the other hand, *all* dirty state must be propagated before ownership transfer can complete, and the ownership transfer process must cryptographically validate that the destination site does indeed hold a correct copy of the state. This is necessary because the downstream site can discard parcel state after control transfer.

As just described, while Horatio is the parcel owner, it must maintain copies of all dirty data that has not yet been committed on the server. In addition, Horatio may optionally retain copies of unmodified parcel data. This allows Horatio to serve as a lookaside cache, supplementing the client's own cache. Cache misses can be serviced from the well-connected Horatio rather than a poorly-connected server.

4.4 Resource Conservation on Horatio

In focusing on Horatio, it is important not to lose sight of the fact that the primary function of a smart phone remains voice communication and texting. If its Horatio functionality drains its battery to the point where the primary function is impacted at a critical moment, the net effect for the user can be quite negative. Striving to reduce Horatio's mobility footprint is therefore an important aspect of our design and implementation. We offload as much resource-intensive work as possible to clients and servers. This is reflected both in the structure of Horatio-client and Horatio-server communication protocols, and in low-level implementation aspects. In this context, a major advantage of a wired client-Horatio communication link such as USB is that it also has the potential to supply energy to the Horatio device.

Offloading work to clients and servers also has a positive impact on performance. Smart phone designs must balance platform capabilities against battery lifetime, and therefore have limited CPU, memory, and storage capacity. Even when the ISR client is a laptop, it is typically much more computationally capable than a smart phone. Further, in ISR scenarios such as Section 2.2.1, laptop battery life tends to be less critical than Horatio battery life. Of course, these considerations have to be balanced against the goal of improving ISR user experience. When necessary, Horatio should be prepared to supply its own resources in order to ensure that the user's needs are met. For example, if Horatio has good connectivity to a server via a cellular data network, it should be prepared to upload modified parcel state rather than requiring the client to do so over a much poorer Internet connection. This might be the case, for example, in the scenario of Section 2.2.2.

4.5 Self-cleaning on Horatio

Despite the high level of trust placed in smart phones, they are also fragile. Since they are small and carried everywhere, they can easily be damaged or lost; they can also run out of energy. In these situations, the effect on the user must be minimized. Therefore, when Horatio holds the most current copy of parcel data, it uses any available connectivity (such as WiFi or 3G) to save that data on the less-fragile server while the user is performing other tasks.

Once a parcel has been checked in to a Horatio device, Horatio immediately begins *self-cleaning*: checking in the parcel to the server using any available wireless connectivity, such as WiFi, WiMax, UWB, or a cellular data network. Because Horatio is a trusted device and the user need not wait for the transfer to complete, it is acceptable for this process to take some time. However, the parcel data remains vulnerable to the loss or destruction of the Horatio device until self-cleaning completes. Optionally, Horatio can retain parcel ownership even after self-cleaning completes; this approach provides maximum resiliency to loss of the Horatio device, while still allowing the next resume site to check out from Horatio rather than from the server. If Horatio possesses both parcel ownership and cached copies of all of the parcel data, the parcel can be checked out from Horatio and resumed on a fresh client that has no Internet connectivity.

If the Horatio device is lost or damaged while Horatio owns the parcel, dirty parcel state stored on the device will be lost. The recovery procedure in this event is the same as that for a lost or damaged ISR client: the user can instruct the ISR server to forcibly make itself the current parcel owner, invalidating modified state held by the previous owner and allowing another client to check out the parcel. This operation will roll back the parcel to its most recent commit on the server. Should Horatio or a client later attempt to check in the invalidated state, the ownership nonce will no longer match that at the server, and the checkin will be rejected.

4.6 Additional Optimizations

Our prototype contains several additional optimizations for improving suspend latency and energy efficiency. These are described in Sections 4.6.1 to 4.6.3 below.

4.6.1 Concurrent Upload from Multiple Sites

As discussed in Section 4.2, the separation of control and data state allows for the concurrent propagation of data state to a server from multiple sources. To reduce the energy demands on Horatio, we take advantage of the fact that any client can continue to transfer data state to a server even after ownership has been transferred to Horatio. After checking in to Horatio and thus relinquishing parcel ownership, an ISR client will attempt to upload modified parcel data directly to the server. Because Horatio maintains a complete copy of the modified data, the user need not trust that the client will complete the upload successfully, but any data uploaded by the client is data that Horatio does not have to expend energy to transmit. Horatio, through its interaction with the Server daemon (see Section 4.7), will validate the cryptographic hashes of data uploaded by clients and ignore invalid or incomplete data, so correctness is preserved even if the client is compromised after the user departs. This optimization strictly improves Horatio's battery utilization: if the client fails to upload dirty data, or maliciously uploads invalid data, Horatio expends no more energy in self-cleaning than if the client did not upload at all.

4.6.2 Memory Image Differencing

OpenISR 0.9.4, upon which the Horatio prototype is based, treats a parcel's memory image as a single, large object. Thus, if a parcel is resumed even for a moment, the entirety of the memory image must be transferred to the server at checkin. However, the ISR server will always have, and Horatio will often have, a copy of the memory image as of the most recent checkin (the "basis image"). We have modified the ISR client in the Horatio prototype so that if the destination of a checkin does possess such a copy, the client computes and transmits only a delta between the basis and current memory images. This improves energy efficiency, suspend latency, and self-cleaning time.

If the client is performing a checkin to the server, the server is responsible for applying the delta to the basis image to produce the new memory image. If the checkin is to Horatio, however, applying the delta is too expensive an operation to be performed on the Horatio device. The delta is therefore saved separately on Horatio, forwarded to the server during self-cleaning, and applied at the server when the parcel is committed. If another client checks out the parcel from Horatio, the basis image plus the delta is sent.

4.6.3 Eager State Propagation

In the design described in Section 4.2, transfer of modified data state to Horatio only begins at checkin, after the end of a user's session. At this point, she must wait for potentially several hundred MB of data to be transferred. While this may be faster than transferring the data to the server over a slow connection, it may still be unacceptably slow. The client therefore opportunistically collects modified disk chunks and memory image state from the parcel while it is running, and speculatively transmits this data to Horatio in the background. Due to this speculation, the state that is collected in this way may not be internally consistent, but it serves to prepopulate the Horatio device with data that is likely to be needed. At checkin, only the final set of differences between the previously-transmitted and the final state are sent to Horatio, minimizing the amount of time that the user must wait. Additionally, the final set of differences fixes any inconsistencies incurred due to the speculation. Also, because of potential update locality in the workload, some of the data transmitted to Horatio may be overwritten before suspend. Eager data transfer thus increases the total amount of data that needs to be transferred, and therefore,

	Openmoko Neo FreeRunner	Nokia N95-8GB	Sandisk Mobile Ultra
Abbreviation	OM	N95	SD
Type	Smart phone	Smart phone	microSD card
Operating System	Linux 2.6	Symbian OS 9.2, S60 rel 3.1	-
Processor	ARM920T 400 MHz	ARM11 332 MHz	-
Memory	128 MB	128 MB	-
Internal Flash	256 MB	128 MB	2 GB
Additional Flash	2 GB microSD	8 GB Internal	-
Connectivity	Full-Speed USB (12 Mbps) Bluetooth 2.0 802.11g, GPRS	Full-Speed USB (12 Mbps) Bluetooth 2.0 802.11g, 3G	Hi-Speed USB (480 Mbps) microSD

Table 3: Portable Devices Used In Horatio Evaluation

Workload Name	Execution Time (s)	Dirty State (MB) Memory	Dirty State (MB) Disk	Workload Description
Email	327.6 (0.5)	16.1 (0.5)	3.4 (0.1)	Use Evolution email client to download, read, and reply.
Word	604.7 (1.1)	40.8 (3.5)	3.3 (0.1)	Use OpenOffice Writer to compose and save a letter.
Photo	803.7(1.3)	24.5(0.1)	3.9(0.0)	Use GIMP image editor to edit photos in an album: remove red-eye artifacts from persons in the photos, clone a person multiple times, and make color and lighting adjustments to photos.
Shop	690.8(2.2)	30.6(0.1)	13.8(0.2)	Shop online for a TV: browse websites of online retailers using Mozilla Firefox, and note prices and technical details with `gedit` text editor.
Podcast	419.0(0.7)	120.4(0.3)	109.3(0.1)	Download a 108 MB MP3 audio podcast using the Rhythmbox media player, and transfer the new podcast to portable music device.
Video	2382.0(0.0)	263.8(5.7)	368.1(0.9)	Download and watch a 30-minute, 378 MB MPEG-4 file.

Results are execution times in seconds and state sizes in MB. Each result is the mean of two measurements. Standard deviations are reported in parentheses.

Table 4: Macrobenchmark Workloads

the amount of energy expended by Horatio. The user can address this concern by connecting the Horatio device to a power source while using the client.

The Horatio design also allows for concurrent eager state propagation, similar to concurrent state propagation that occurs at suspend time (see Section 4.6.1). In this case, if ample bandwidth between the ISR client and server exists, the client may choose to eagerly propagate state modifications to the server, rather than to Horatio. In fact, given the broad possible range of link performance combinations between the client-Horatio and client-server links, the ISR client can optimize eager state propagation by selectively choosing to which target (Horatio, ISR server, or both) to eagerly propagate the state modifications. Our current system allows the user to decide.

4.7 Implementation

The Horatio prototype implementation consists of three separate components: the Client daemon, the Horatio Phone daemon, and the Server daemon, running respectively on the client, Horatio device, and server. The Client and Horatio Phone daemons are responsible for performing state and ownership transfers between their respective devices. The Horatio Phone daemon performs self-cleaning, while the Client daemon performs concurrent upload, by communicating with the Server daemon. The Server daemon, in turn, interacts directly with the ISR server to assist the other daemons with these tasks.

The Horatio prototype currently runs on two smart phone platforms, the Openmoko Neo FreeRunner and the Nokia N95. Specifications for these devices are given in Table 3. The FreeRunner is Linux-based, and supports GPRS and WiFi wireless connectivity; the N95 is Symbian S60-based, and supports 3G and WiFi.

Most components of Horatio are written in Python version 2 for portability. Performance-critical sections of the state transfer code are implemented in C and compiled natively for each platform. Use of the Symbian S60 Open C libraries on the N95 allowed the same C source code to be used for both smart phones. Horatio also required changes to OpenISR client and server code. The client was modified to initiate the Client daemon. The server was modified to support applying memory image deltas, as described in Section 4.6.2, and to support ownership transfer between the client and Horatio.

5. EVALUATION

Our experimental evaluation of the Horatio prototype addresses four questions:

- How much does Horatio improve user experience (Section 5.2)?
- How effective is self-cleaning in reducing the vulnerability of a Horatio device (Section 5.3)?
- What is the impact of Horatio on a user's mobility footprint (Section 5.4)?
- How effective is eager state propagation in reducing suspend latency (Section 5.5)?

5.1 Experimental Methodology and Setup

We use two types of experiments in evaluating Horatio. The "microbenchmark" experiments use synthetically generated parcel state. The "macrobenchmark" experiments use parcel state generated by a set of scripted workloads.

5.1.1 Microbenchmarks

Each microbenchmark is based on an initial parcel configured with 512 MB of RAM and a 4 GB disk. For each set of measurements, we vary the amount of dirty data state that must be transferred by synthetically generating a predetermined amount of incompressible state and updating the par-

	Dirty State Size			
Horatio Device	1 MB	10 MB	100 MB	500 MB
ISR-1 (No Horatio)	1433.5(6.5)	1487.5(0.5)	2118.0(3.0)	4936.0(15.0)
N95-WiFi	39.0(2.5)	68.7(2.4)	301.3(5.0)	1239.2(27.5)
OM-WiFi	32.7(2.1)	48.0(0.8)	260.3(5.3)	1040.0 (8.6)
N95-USB	29.3(4.4)	44.0(2.5)	142.0(1.3)	625.0 (4.3)
SD-USB	23.3(0.5)	25.3(0.5)	40.3(1.3)	136.0 (2.2)

Results are suspend times in seconds and are the mean of six measurements. Standard deviations are reported in parentheses. ISR-1 represents the base ISR case over a 1Mbps WAN link.

Table 5: Microbenchmark Suspend Results.

	Dirty State Size				
Horatio Device	0 MB	1 MB	10 MB	100 MB	500 MB
ISR-1 (No Horatio)	281.5(1.5)	-	-	-	-
N95-WiFi	-	397.7(2.5)	409.3 (2.3)	507.5(6.0)	951.2(20.1)
OM-WiFi	-	292.3(1.7)	314.7(12.0)	372.0(2.2)	692.0 (2.2)
N95-USB	-	226.0(3.2)	237.0 (6.7)	283.8(0.7)	520.8 (2.5)
SD-USB	-	34.7(0.5)	34.3 (0.9)	35.7(0.5)	51.3 (0.9)

Results are resume times in seconds and are the mean of six measurements. Standard deviations are reported in parentheses. ISR-1 represents the base ISR case over a 1Mbps WAN link.

Table 6: Microbenchmark Resume Results.

cel disk and memory images. We distribute the dirty state equally between memory and disk, e.g., 100 MB of dirty state would be divided between 50 MB of dirty memory state and 50 MB of dirty disk state.

During suspend and resume, there is a fixed amount of base state that must be transferred along with the dirty state. For suspend, the amount is 8 MB and consists of the control state items (keyring, configuration file, and nonce) that must be transferred to Horatio along with the data state. For resume, the amount is 175 MB consisting of the control state items and the initial VM memory image. This image must be copied to the client as the base memory image to which the dirty state memory diffs are applied. In some situations, a client may already possess a cached copy of the base memory image. In our experiments, we assume the worst case for Horatio and clear the client cache after each run. In future, we plan to demand fetch the memory image, just as we demand fetch VM disk state today.

5.1.2 *Macrobenchmarks*

Our macrobenchmarks consist of six workloads, lasting between three and forty minutes, that exemplify activities commonly performed by desktop users. Table 4 summarizes these workloads. Each workload uses the same parcel, in the same state: a Debian Linux 5.0 guest with 512 MB of RAM and an 8 GB disk, containing all of the applications needed to run the workloads.

5.1.3 *Hardware Setup*

Our evaluation uses three different Horatio devices: *(i)* an Openmoko Neo FreeRunner, *(ii)* a Nokia N95, and *(iii)* a USB-connected flash storage card. Specifications for these devices are given in Table 3. Device *(iii)* does not meet all of the requirements for Horatio, since it cannot perform self-cleaning. We include it because it represents a case that neither the N95 nor FreeRunner support: USB 2.0 in Hi-Speed (480 Mbps) mode. Since new models of smart phones are starting to support both microSD and Hi-Speed USB, it allows us to observe the expected suspend and resume latency for Horatio on emerging devices.

Our ISR client PC is a Dell Optiplex 755 with a 2.33 GHz Core 2 Duo CPU, 3 GB of RAM, a 250 GB Serial ATA disk at 7200 RPM, and support for Hi-Speed USB. The ISR server is a Dell SMP server with dual 2.8 GHz Pentium 4 Xeon processors, 1 GB of RAM, a 32 GB Fast-Wide SCSI disk at 10,000 RPM, and a 1 Gbps Ethernet connection. We consider two different client-Horatio interconnects: 802.11g WiFi and USB. We evaluate WiFi for the FreeRunner and N95 smart phones, and USB for the N95 phone and the USB-connected microSD card. We evaluate Horatio's self-cleaning performance over the Internet via both WiFi and the AT&T Wireless 3G network. The 3G network has advertised upload speeds between 500 Kbps and 1.2 Mbps, and download speeds between 700 Kbps and 1.7 Mbps. For the WiFi experiments, Horatio connects to the Internet via the Linksys 802.11g access point through a 1 Mbps residential broadband Internet connection. The ISR client and server are also connected over the same residential link.

Throughout Section 5, we refer to a particular combination of Horatio device and connectivity via a two-part abbreviated name. The first part of the name is the device type, using the abbreviations given in Table 3. The second part is the relevant connectivity. For example, if a suspend-time measurement refers to "N95-WiFi," it refers to Nokia N95 using WiFi to communicate with the client. For a self-cleaning experiment, the same name would refer to an N95 using WiFi to the server.

5.2 Improvement in User Experience

5.2.1 *Microbenchmarks*

We first evaluate Horatio's suspend and resume times over WiFi and USB, using synthetically generated dirty state as described in Section 5.1.1. Table 5 presents the resulting suspend latencies with 1 to 500 MB of dirty state, and Table 6 presents the resume times. With smaller amounts of dirty state, suspend and resume latencies are dominated by the overhead of transferring the fixed parcel state. As the dirty state size increases, this effect is reduced. We also compare against standard ISR (without Horatio) over a 1 Mbps residential Internet connection.

In the suspend case, Horatio shows a substantial performance improvement over standard ISR. For resume, standard ISR performs better than both Horatio WiFi cases.

The primary reason for this is that these experiments include a fixed amount of dirty state at Horatio, which must be propagated back to the client during resume from Horatio. A resume from the ISR server implies clean state, since all state updates must have been sent back to the server during the last suspend. Therefore, when resuming from the ISR server, only the memory image and control state is propagated to the client. By comparing the base ISR 0MB case to the Horatio with 1MB of dirty state cases (column 2 in Table 6), we observe that both USB cases exhibit reduced resume latencies. Since these cases are dominated by the memory image and control state transfers, we can see that a clean Horatio resumes faster than the base ISR case.

On the FreeRunner and N95, suspend and resume latencies are limited by the USB speeds of the Horatio device. At the higher transfer rates seen with the microSD card, however, the bottleneck becomes the generation and application of memory images on the client PC. In addition, for both suspend and resume times, observed performance with the FreeRunner is consistently better than that obtained with the N95. Further experiments show that the FreeRunner exhibits better WiFi throughput and better write performance to internal storage: the FreeRunner achieves 870 KB/s and 2.03 MB/s, respectively, while the N95 achieves only 690 KB/s and 688 KB/s.

Finally, we note that as the state size passes 100 MB, the observed suspend and resume latencies over WiFi start to approach the limits of what a user might be willing to withstand. Observed latencies over USB are substantially better. The N95 exhibits adequate performance even with only Full-Speed USB, while the microSD card with Hi-Speed USB exhibits latencies under a minute for all but the 500 MB suspend operation.

5.2.2 *Macrobenchmarks*

In this section, we evaluate suspend and resume times experienced with Horatio using the workloads described in Section 5.1.2. The testbed for these experiments consists of an ISR client and server, plus the Horatio device. The client is a desktop system with a 2.66 GHz Intel Core 2 Duo and 2 GB of RAM. Its software consists of Debian Linux 5.0 with kernel 2.6.26, and the VMware Player 2.5.0 virtual machine monitor. The server is a 3 GHz Intel Pentium 4 host with 2 GB of RAM, running Ubuntu Linux with kernel 2.6.27. The two hosts communicate through a `netem`-emulated WAN link with bandwidths of 1 Mbps and 10 Mbps, and a round trip time (RTT) of 20 ms. In today's Internet, 10 Mbps represents a highly favorable connection. We evaluate Horatio suspend and resume performance over WiFi using the Nokia N95, and over USB using the microSD card.

The results of these experiments are presented in Figure 5. The figure includes workload execution times because they represent productive work from a user's viewpoint — in the ideal case, workload execution time would dominate suspend and resume times and thus result in an all-black bar. As the figure shows, Horatio improves both suspend and resume times compared to standard ISR, particularly when Horatio is connected to the client over USB. The improvement is the most dramatic on the 1 Mbps link. If Jill were to watch a thirty-minute video at the coffee shop with a 1 Mbps connection to the ISR server, her suspend time would be more than one and a half hours with standard ISR, but just over five minutes with Horatio over USB.

5.3 Effectiveness of Self-Cleaning

We evaluate the effectiveness of Horatio's self-cleaning functionality by determining the window of time during which parcel data stored on Horatio would be vulnerable to data loss — that is, the amount of time required for self-cleaning. Intuitively, a user would wish to minimize this time, but a vulnerability time of minutes or hours, rather than days, is likely to be acceptable.

5.3.1 *Microbenchmarks*

To perform the self-cleaning microbenchmarks, we use the experimental setup described in Section 5.1. The Horatio device self-cleans 4.5 MB of fixed state and between 1 MB and 100 MB of dirty state, uploading to the ISR server over WiFi or 3G wireless connectivity. We evaluate self-cleaning over WiFi using both the Nokia N95 and Neo FreeRunner. 3G connectivity is evaluated using only the N95, since the FreeRunner does not support 3G.

	Dirty State Size		
Horatio Device	1 MB	10 MB	100 MB
N95-WiFi	36.3 (0.9)	97.0 (0.0)	869.0 (15.1)
OM-WiFi	13.0 (0.0)	82.3 (0.5)	775.0 (0.8)
N95-3G	152.7 (1.3)	477.7 (14.4)	3848.3 (102.8)

Self-cleaning times (in seconds) are given as the mean and standard deviations of three measurements (in parentheses).

Table 7: Self-Cleaning Time (Microbenchmarks)

As Table 7 shows, self-cleaning times are reasonable in all cases. The longest times are observed for the N95-3G experiments. With 100 MB of dirty state, the N95 was able to self-clean over 3G in approximately one hour. Use of 802.11g substantially improves self-cleaning performance, reducing this worst-case time to just under 15 minutes.

5.3.2 *Macrobenchmarks*

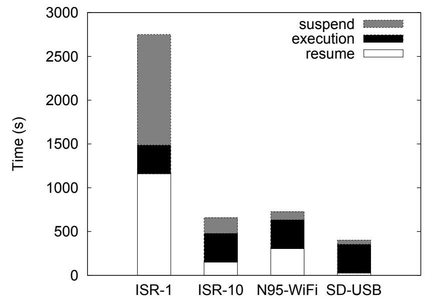
In this section, we evaluate self-cleaning using the workloads described in Section 5.1.2, over both WiFi and 3G wireless connectivity. The Horatio device transfers 4.5 MB of fixed state, plus the amount of dirty state generated by the workload, to the ISR server described in Section 5.1.3. The N95 smart phone is used as the Horatio device.

Workload	N95-WiFi	N95-3G
Email	213.3 (14.0)	739.3 (5.3)
Word	953.0 (9.4)	4353.5 (90.5)
Photo	839.3 (19.0)	3381.0 (129.0)
Shop	1103.0 (50.3)	4830.0 (313.0)
Podcast	2199.7 (176.3)	*6398*
Video	*8034*	*23665*

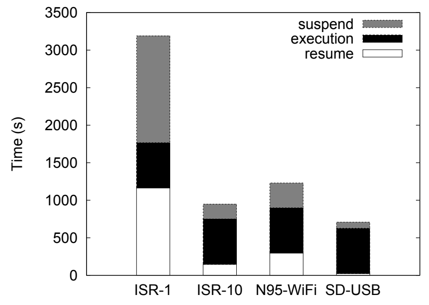
The self-cleaning times (in seconds) are given as the mean of three measurements. Standard deviations are reported in parentheses. Results reported in *italics* are estimates based upon measured self-cleaning rates.

Table 8: Self-Cleaning Time (Macrobenchmarks)

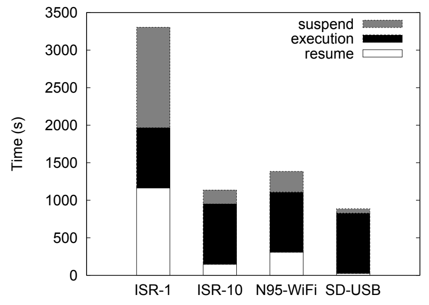
As Table 8 shows, the results are comparable to those in Section 5.3.1. Most cases complete the self-cleaning process in under an hour, and all but two complete in under two hours. The worst case self-cleaning time takes an estimated 6.5 hours to complete; this is for the most data-intensive



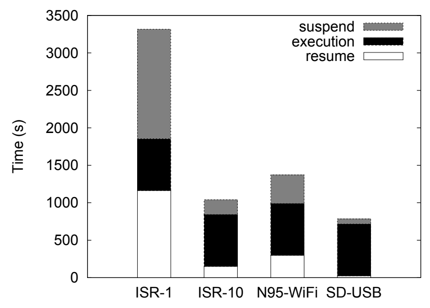
(a) Email



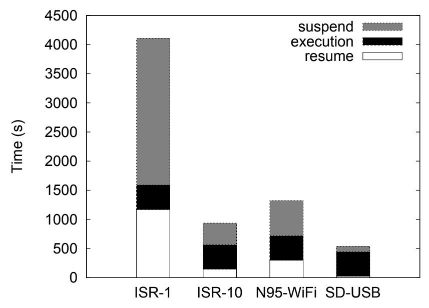
(b) Word



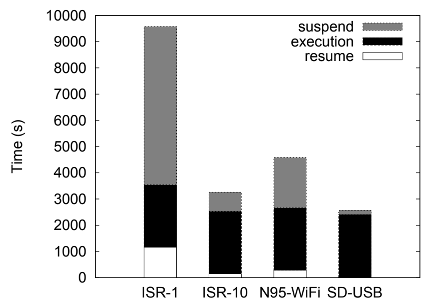
(c) Photo



(d) Shop



(e) Podcast



(f) Video

The figure shows that Horatio reduces suspend and resume times experienced by the user significantly over normal ISR. The ideal case is an all black bar. The ISR-1 and ISR-10 bars represent normal ISR with an emulated WAN link of 1Mbps and 10Mbps, respectively. Results are the mean of three measurements and standard deviations were below 19% in all cases. The scales of the graphs differ to improve the presentation.

Figure 5: Suspend and Resume Overheads (Macrobenchmarks)

workload, which generates over 700 MB of dirty state. Although the self-cleaning times are longer for the two download workloads, we observe that users are likely to place substantially more value in personally-generated data, such as their word processing documents and spreadsheets, than they would in downloaded data such as multimedia files. This is because when lost, most downloaded content can be fetched again. Therefore, Horatio performs very well under the most important workloads.

Finally, in all cases there is a substantial performance benefit to using WiFi over 3G. Based upon the observations from both the micro– and macrobenchmark experiments, we conclude that self-cleaning can successfully reduce the window of data vulnerability to an acceptable duration.

5.4 Impact on Mobility Footprint

In this section, we present an evaluation of the energy costs associated with the suspend and resume operations. We also evaluate the energy cost of self-cleaning. In actual use, the user should not have to pay the suspend and resume costs, since the Horatio device should be able to charge its battery while connected to a PC client. However, it is possible that an external power source may not be available for Horatio while the parcel is running.

We use the microbenchmark described in Section 5.1.1 to determine the energy demands for three operations: *(i)* suspend, *(ii)* resume, and *(iii)* self-cleaning. We report the amount of energy consumed by the N95 smart phone during each operation, along with the percentage of battery utilization this represents. For the suspend and resume operations, we conduct the experiments over both 802.11g WiFi and USB. For self-cleaning, we perform the measurements for WiFi and 3G. Energy consumption is measured using the Nokia Energy Profiler utility [3].

Table 9 shows the energy consumed during each measurement, and the percentage of battery power utilized. We calculate the percentage from our energy measurements and the battery capacity as reported in the device's specifications. Although we assume a linear relationship between energy consumption and battery lifetime, it has been shown that batteries typically exhibit non-ideal properties [11], and the relationship between energy consumption and battery lifetime is typically non-linear. To ensure consistent measurements, we fully charged the battery prior to each experimental run. Still, it is likely that our battery percentage calculations are affected by this non-ideality, and are only included to provide an alternative representation of the energy costs of Horatio. As in the previous microbenchmark experiments, we vary the synthetic dirty state from 1 MB to 500 MB for suspend and resume, and from 1 MB to 100 MB for self-cleaning.

As the results show, connecting Horatio via USB is more than twice as efficient than via WiFi during suspend, and more than five times as efficient during resume. Two factors contribute to this result. First, WiFi communication inherently requires more energy than USB. Second, when connecting Horatio over USB we are able to treat it as a mass-storage device rather than a networked host, allowing us to shift all of the suspend and resume computation to the client PC. The USB connection therefore allows us to minimize the work that must be done by the Horatio device during suspend and resume.

For self-cleaning, WiFi is more efficient than 3G by at least a factor of five. This is due to the higher throughput of WiFi:

		Dirty State Size			
Operation	Horatio Device	1 MB	10 MB	100 MB	500 MB
Suspend	N95-WiFi	27.5 (1.0)[0.2%]	71.2 (3.9)[0.4%]	400.1 (1.4) [2.5%]	1788.8 (8.9)[11.0%]
Suspend	N95-USB	12.0 (2.5)[0.1%]	31.3 (0.8)[0.2%]	146.8 (3.8) [0.9%]	608.5(14.2) [3.7%]
Resume	N95-WiFi	507.1(40.6)[3.1%]	612.8 (1.0)[3.8%]	756.3 (15.5) [4.6%]	1455.7(33.4) [8.9%]
Resume	N95-USB	95.5 (1.6)[0.6%]	96.8 (1.7)[0.6%]	120.0 (1.2) [0.7%]	226.6 (0.5) [1.4%]
Self-Clean	N95-WiFi	35.7 (0.7)[0.2%]	102.6 (1.4)[0.6%]	915.6 (3.2) [5.6%]	-
Self-Clean	N95-3G	180.6 (4.3)[1.1%]	565.1(13.4)[3.5%]	4552.7(107.8)[27.9%]	-

Results above are the mean of three measurements. Results expressed in Joules. Standard deviations are reported in parentheses. Results in brackets expressed as percentage of battery life.

Table 9: Horatio Energy Consumption (Microbenchmarks)

self-cleaning times over WiFi are also shorter than over 3G by a factor of five on average, enabling the Horatio device to power its transmitter for much less total time. These results clearly motivate the opportunistic use of WiFi. It would be especially helpful in the case of large self-cleaning transfers, since self-cleaning 100 MB over 3G consumes nearly 28% of the device's battery capacity. In contrast, a similar operation over WiFi consumes less than 6%. Overall, it is clear that Horatio does increase the user's mobility footprint with respect to energy consumption. However, with opportunistic use of WiFi this increase can be kept small, and is offset by the fact that the user is not required to carry any other devices to gain this benefit.

5.5 Eager State Propagation

To further reduce the suspend latency after a user completes her ISR session, our implementation includes a mechanism for eager state propagation as described in Section 4.6.3. The goal of this section is to quantify the benefit and cost of this optimization in terms of the amount of dirty state transferred from a client to Horatio. To accomplish this, we use a set of *state generation profiles*, which are traces that characterize the updates performed to a parcel's state during an ISR session. We gathered the state generation profiles on four of the six workloads described in Section 5.1.2.

To capture a state generation profile for a specific workload, we execute the workload in a parcel while a background task takes periodic snapshots of the parcel's memory and disk state. We pause the session during each snapshot to ensure consistency. The resulting set of snapshots and accompanying traces allow us to deterministically replay the state updates produced by the session.

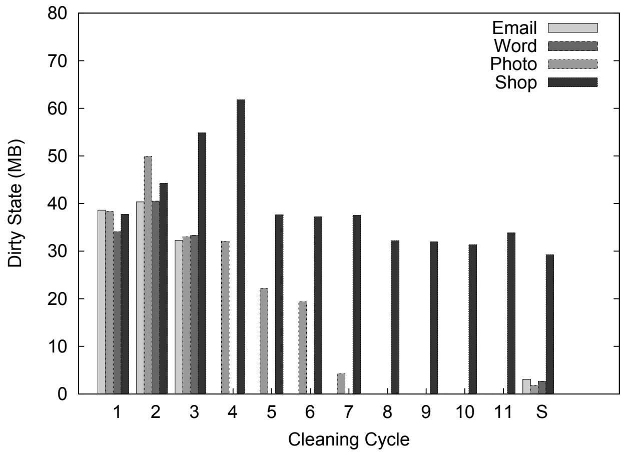
Workload	Suspend State	Eager State	Lazy State	Cleaning Cycles
Email	3.1	129.6	19.5	3.0
Word	1.7	220.8	44.1	3.3
Photo	1.6	199.1	28.4	6.7
Shop	29.3	485.7	44.4	11.0

Results presented are state sizes in MB and are the mean of three measurements. The maximum standard deviations for Suspend State, Eager State, and Cleaning Cycles were all under 6% of the corresponding mean.

Table 10: Horatio Eager State Propagation Results

To evaluate eager state propagation, we replay our state generation profiles on a real parcel while the eager propagation mechanism transfers dirty memory and disk state to Horatio in the background. We use the client PC described in Section 5.1.3 and the microSD card as the Horatio device.

Table 10 presents the results. The second and third columns give the amount of state (in MB) transferred at and before



The distribution of state updates (in MB) for each workload during each periodic cleaning cycle. A cleaning cycle starts approximately one minute after the end of the preceding cycle. Suspend time is marked by an "S" on the horizontal axis.

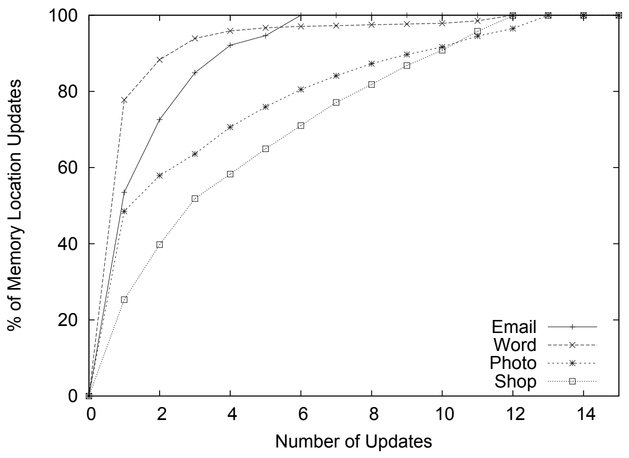
Figure 6: Workload Dirty State Generation

suspend, respectively, and correspond to the benefit and cost associated with eager propagation. The fourth column gives the amount of state that would be transferred at suspend if eager state propagation were not used. Finally, the last column gives the number of eager transfer cycles performed by the client over the course of the session.

The table shows that the benefit of eager state propagation is very high for the four workloads tested. For the first three, the amount of dirty state transferred at suspend is less than 5 MB. The fourth workload contains a large number of state updates late in the session, which do not have time to propagate to Horatio before suspend, but are still less than the amount transferred without eager propagation. Finally, the ratio of the fourth column to the second column is a good indication of the user-perceived improvement in performance. In all four workload cases, a client would perceive a performance benefit with Horatio.

As to cost, the total data transferred due to eager state propagation is roughly $5 - 7$ times that occurring without eager transfer, except for the Email workload, where the difference is a factor of 11. From these results, we conclude that eager state transfer is worth the cost: the amount of state that must be transferred during suspend is dramatically reduced, in exchange for a reasonable increase in the amount of data to transfer. We do not evaluate the energy costs of eager state propagation since we assume that local power (e.g., through USB, or wall socket) is available to Horatio when state is being transferred to/from an ISR client.

Figure 6 presents the distribution of state updates (in MB) for each of the four workloads for each periodic cleaning cy-



CDF of the number of updates to each 4 KB memory page, for all memory updates within each workload.

Figure 7: Update Locality

cle. In the figure, one run from each workload is shown, so that the results align directly with cleaning cycles, as compared to the average results presented in Table 10. The right-most set of bars represents the state transferred at suspend time, which is marked by an "S" on the horizontal axis. These results illustrate the fact that the rate of data generation by typical user workloads underutilizes the available write capacity (approx. 360MB/min for microSD). Out of the four shown, the Shop workload generates the largest amount of updates, yet still only approaches 17% link utilization. Shop also performs a large number of updates at suspend time, relative to the other three workloads, and this accounts for the differences shown in Table 10.

Figure 7 presents a CDF that illustrates the effects of update locality during each of the four user workloads. We measure this by periodically summing, at 1 minute intervals, the number of updates to each memory location during workload execution. For each workload, the graph plots the CDF of the number of updates to a 4KB memory page, for all memory updates. The Word workload exhibits the least amount of update locality with approximately 77% of updated memory pages only being updated 1 time and 94% updated 3 or less times. The Email workload is similar to the Word workload with 53% updated 1 time and 95% updated 5 or less times. The Online workload exhibits the largest amount of update locality with only 25% of updated memory pages being updated 1 time and 95% being updated 11 or less times. Finally, the Photo workload is similar to the Online workload with 48% updated 1 time and 95% updated 12 or less times. From these results, we conclude that by adaptively adjusting the eager propagation algorithm to account for observed update locality during a session, the costs of eager state propagation may be further reduced.

6. DISCUSSION AND FUTURE WORK

6.1 Smart Phones as Horatio Platforms

While our experimental results clearly indicate that the current smart phone technology is already adequate to support Horatio, there are several ways in which this technology could better serve it if improved. Connectivity and storage performance are the two most critical factors on which Horatio's performance ultimately depends. Incorporating high-speed flash storage should become standard in order to allow smart phones to benefit from most recent flash memory technology. Supporting high-performance implementations of the latest interconnects such as USB 2.0, WiMAX and others allows phones to choose the most effective way to perform self-cleaning at any given location. The phone's network stack also needs to be improved to fully exploit existing technologies, such as WiFi and USB. For example, while the Openmoko and the N95 both provide 802.11g, which is rated at 54 Mbps, the bandwidth in our experiments maxed out just under 6.8 Mbps. Fortunately, Horatio's needs align well with current trends towards transforming the smart phone into a personal mobile multimedia entertainment platform.

6.2 VM State Size

In Section 4 of the paper, we describe the storage capability requirements for smart phones. Since current smart phones support 16GB or more of storage via microSD, it is realistic to assume that they provide ample capacity for VM state modifications and caching. However, Horatio also supports operation when fully disconnected from the server. In this mode, the entire VM state must be copied to Horatio in advance of disconnection; this may require more than 16GB. We expect current disk growth trends to continue, and newer smart phones to support even larger storage capacity.

6.3 Impact of Network Connectivity

As discussed in Section 4, we assume strong connectivity between an ISR client and Horatio, while we allow for the broadest range of connectivity options between Horatio and the ISR server, and between the ISR client and ISR server. In practice, both ISR and Horatio must be able to handle and recover from poor connectivity conditions, including disconnection, when they occur. Poor connectivity between an ISR client and server may cause the user's session to experience a performance reduction, or in the case of disconnection, may block the user session while the ISR client waits for reconnection in order to fetch missing state from the ISR server. Poor connectivity between Horatio and an ISR server will impact self-cleaning performance, possibly even pausing it in the event of a disconnect until the link has been reestablished. Once reestablished, self-cleaning can be resumed from where it left off.

Figure 5 shows that as the bandwidth between an ISR client and server increases from 1 Mbps to 10 Mbps, resume and suspend delays are reduced. This mitigates the need for Horatio. Although Internet bandwidths will grow over time, so will average VM sizes. It is difficult to predict the net effect of these factors.

6.4 Resume Location Prediction

Resume latency can be further reduced if the identity of the next resume site is known or can be correctly predicted. ISR state can be proactively transferred to that site, effectively warming ISR cache state. Potential sources of knowledge for accurate prediction include a smart phone's localization device such as GPS, and personal calendar information. We plan to explore this in our future work.

6.5 Horatio User Interface

We believe that a user's experience with Horatio can be further improved by providing a richer GUI. Such an interface may provide users feedback and control over various Horatio operations. An obvious candidate is the timing of the suspend operation, where Horatio might be able to allow users to indicate an upper bound for the suspend latency. This can potentially be enforced if Horatio's client daemon is allowed to take some action for the user (warning, slow down or stop) when the amount of dirty state to be transferred exceeds a threshold calculated based on the user's suspend time requirements and estimated transfer bandwidth.

7. CONCLUSION

We introduced Horatio, a system that leverages the storage and Internet connectivity of smart phones to improve the experience of ISR users in environments with poor wide-area network bandwidth. Horatio treats smart phones as trusted personal assistants that serve as self-cleaning portable caches for VM state. Horatio reduces suspend latency by saving VM state to the phone rather than directly to the server; similarly, users can resume their session from Horatio. To reduce vulnerability to loss or damage, Horatio opportunistically uses the phone's network connectivity to asynchronously propagate modified VM state to ISR servers (self-cleaning).

Our experiments with the Horatio prototype running on the Nokia N95 and the Openmoko Neo FreeRunner smart phones show that Horatio reduces resume and suspend latencies by up to 98% and 97%, respectively. For example, Horatio reduces the suspend latency from 38.3 minutes (for a demanding workload) or 3.3 minutes (for a light workload) down to 1.2 minutes. While the performance of Horatio on existing phones is adequate, our experiments also show that improvements in the phones' protocol stacks and software environments could further improve Horatio's performance by an order of magnitude.

8. ACKNOWLEDGEMENTS

We thank our shepherd Ed Nightingale and the anonymous reviewers for their valuable comments and suggestions. This work is supported in part by the National Science Foundation through grant numbers CNS-0520123 and CNS-0509004. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, Carnegie Mellon University, Rutgers University, or the University of Toronto. *Internet Suspend/Resume® (ISR)* and *OpenISR®* are registered trademarks of Carnegie Mellon University.

9. REFERENCES

[1] GoToMyPC home page. `http://www.gotomypc.com`, 1997-2008.

[2] MojoPac home page. `http://www.mojopac.com`, 2008.

[3] Nokia Energy Profiler home page. `http://www.forum.nokia.com/Resources_and_Information/Explore/Development_Process_and_User_Experience/Power_Management/Nokia_Energy_Profiler_Quick_Start.xhtml`, 2008.

[4] Balan, R. K., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H.-I. The Case for Cyber Foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop* (Saint-Emilion, France, July 2002).

[5] Cáceres, R., Carter, C., Narayanaswami, C., and Raghunath, M. Reincarnating PCs with Portable SoulPads. In *MobiSys '05: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services* (Seattle, WA, 2005).

[6] Garriss, S., Cáceres, R., Berger, S., Sailer, R., van Doorn, L., and Zhang, X. Trustworthy and Personalized Computing on Public Kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (Breckenridge, CO, June 2008).

[7] Kim, M., Cox, L. P., and Noble, B. D. Safety, Visibility, and Performance in a Wide-Area File System. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies* (Monterey, CA, January 2002).

[8] Kistler, J., and Satyanarayanan, M. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[9] Kozuch, M., Satyanarayanan, M. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, June 2002).

[10] Lagar-Cavilla, H. A., Tolia, N., de Lara, E., Satyanarayanan, M., and O'Hallaron, D. Interactive Resource-Intensive Applications Made Easy. In *Proceedings of the 8th ACM/IFIP/USENIX International Middleware Conference* (Newport Beach, CA, Nov. 2007).

[11] Martin, T. L., and Siewiorek, D. P. Non-ideal battery properties and low power operation in wearable computing. In *ISWC '99: Proceedings of the 3rd IEEE International Symposium on Wearable Computers* (Washington, DC, USA, 1999), IEEE Computer Society, p. 101.

[12] Ravi, N., Narayanaswami, C., Raghunath, M., and Rosu, M.-C. Securing pocket hard drives. *IEEE Pervasive Computing 6*, 4 (2007), 18–23.

[13] Satyanarayanan, M., Gilbert, B., Toups, M., Tolia, N., Surie, A., O'Hallaron, D. R., Wolbach, A., Harkes, J., Perrig, A., Farber, D. J., Kozuch, M. A., Helfrich, C. J., Nath, P., and Lagar-Cavilla, H. A. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing 11*, 2 (2007), 16–25.

[14] Surie, A., Perrig, A., Satyanarayanan, M., and Farber, D. J. Rapid Trust Establishment for Pervasive Personal Computing. *IEEE Pervasive Computing 6*, 4 (2007).

[15] Tolia, N., Harkes, J., Kozuch, M., and Satyanarayanan, M. Integrating Portable and Distributed Storage. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (San Francisco, CA, March 2004).

[16] Tolia, N., Kaminsky, M., Andersen, D. G., and Patil, S. An Architecture for Internet Data Transfer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)* (San Jose, CA, May 2006).

[17] Tolia, N., Andersen, D., Satyanarayanan, M. Quantifying Interactive Experience on Thin Clients. *IEEE Computer 39*, 3 (Mar. 2006).