Pervasive Personal Computing in an Internet Suspend/Resume System

The Internet Suspend/Resume model of mobile computing cuts the tight binding between PC state and PC hardware. By layering a virtual machine on distributed storage, ISR lets the VM encapsulate execution and user customization state; distributed storage then transports that state across space and time. This article explores the implications of ISR for an infrastructure-based approach to mobile computing. It reports on experiences with three versions of ISR and describes work in progress toward the OpenISR version.

Mahadev Satyanarayanan, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, and David J. Farber
Carnegie Mellon University

Michael A. Kozuch and Casey J. Helfrich
Intel Research Pittsburgh

Partho Nath
Pennsylvania State University

H. Andrés Lagar-Cavilla
University of Toronto

Portable computers have been the driving technology behind mobile computing since the early 1990s. Today, the phrase *mobile computing* is almost synonymous with the use of laptop and handheld computers. However, the plummeting cost of hardware suggests that the pervasive computing infrastructure might some day eliminate the need to carry such hardware.

In this article, we describe a new approach to mobile computing that embraces this opportunity – specifically with the Internet Suspend/Resume (ISR) system that emulates the suspend/resume capability of laptop hardware. Rather than carrying hardware, we might find and use hardware transiently at any location. Imagine a world where coffee shops, airport lounges, dental and medical offices, and other semipublic spaces provide hardware for their clientele. Even the foldout tray at every seat in an airplane or commuter train could be a laptop. In that world, users could travel hands-free yet make productive use of slivers of time anywhere such infrastructure is available. This technical capability could inspire new business models, centered on meeting customer demand for trustworthy computing hardware at any time and place and on preserving PC state on servers.

Why is infrastructure-based mobile computing attractive? There are obvious advantages, such as traveling with less luggage, simplifying security screening in a post-9/11 world, and being able to get

1089-7801/07/$25.00 © 2007 IEEE

Think Wallet, Not Swiss Army Knife

The Internet Suspend/Resume (ISR) system offers a fundamentally different way of thinking about mobile computing and about the resources that we need to carry. The current design philosophy for mobile devices resembles a Swiss Army knife approach: cram as much functionality as possible into a single device. Unfortunately, as anyone who has used a Swiss Army knife knows, its value as survival gear is much higher than its value in everyday use. If you are stranded far from civilization, the many functions of a Swiss Army knife (such as knife, fork, can opener, corkscrew, screw driver, tooth pick, and so on) can be a life saver. But each function is suboptimally implemented relative to a full-sized implementation. For example, you would much rather use a full-sized screw driver, if available, than the small, hard-to-use one in a Swiss Army knife. The same logic applies to that device's other functions.

Contrast this with the different design philosophy of a wallet. Virtually every adult carries a wallet. The contents vary, but they typically include things like cash, credit cards, ID cards, and so on. Far from civilization, a wallet is useless. You might die of starvation or thirst in the wilderness even with a fat wallet; a Swiss Army knife would be much more useful in those circumstances. In the context of civilization, however, a wallet is more useful. With cash or a credit card, you can obtain anything you need, when and where you need it.

Thus, we can view a wallet as a device that helps transform generic infrastructure into highly personalized services. This observation leads to a different design philosophy for mobile devices. Rather than cramming direct functionality into the device, we put indirect functionality in it. This indirect functionality leverages the external environment to provide direct functionality on demand. Carrying a Trust-Sniffer device integrated with USB storage for look-aside caching brings us close to the model of carrying a wallet rather than a Swiss Army knife. Such a device could even be the size of a credit card and fit into your wallet!

work done at unexpected times and locations even if you didn't have the foresight to bring your laptop. More fundamentally, infrastructure-based mobile computing can liberate us from the rigid design constraints of portable hardware. For a given cost and level of technology, considerations of weight, power, size, and ergonomics exact a penalty in attributes such as processor speed, memory size, and disk capacity. Although mobile elements will improve in absolute ability, they will always be resource-poor relative to static elements of the same vintage. Furthermore, the dependence on a finite energy source and the need to monitor remaining energy is an ongoing distraction for mobile users.

Liberating Personal Computing

The world we envision will retain the user customization aspect of personal computing. However, computers themselves will become a ubiquitous resource, much like light at the flip of a switch, water from a faucet, or the air we breathe. (See the "Think Wallet, Not Swiss Army Knife" sidebar for an extended discussion of this idea.) On demand, any Internet-connected computer could temporarily become your personal computer. Any machine will be able to acquire a user's unique customization and state from a server. When a user is done, his or her modified state is erased from that machine and returned to the server. Loss, theft, or destruction of the machine will become only a minor inconvenience, not a catastrophic event.

To attain this vision, we need to solve at least three difficult technical problems:

- *Provide efficient on-demand access to a user's entire personal computing environment.* Today, users who carry a portable computer are assured of seeing exactly the same set of personal files, operating system, customizations, and so on everywhere they go. Precise and complete recreation of this familiar context is the key to low user distraction in any future mobile computing model. Just providing access to a user's personal files or to application customizations won't suffice.
- *Ensure resilience to Internet vagaries.* Today, users working with local data on portable computers are unaffected by network quality. They aren't impacted by unpredictable bandwidth and latency or by occasional failures. This defines the standard against which consumers will judge new models of mobile computing. Users must perceive crisp, stable, interactive performance even under conditions of high network latency and network congestion. The building blocks of modern-day user interfaces such as scrolling, highlighting, and popup menus all assume a tight feedback loop between users and their applications. Only a thick-client solution, in which the application executes close to the user, can support this tight feedback loop when network connectivity is poor.[1] In the extreme case of network discon-

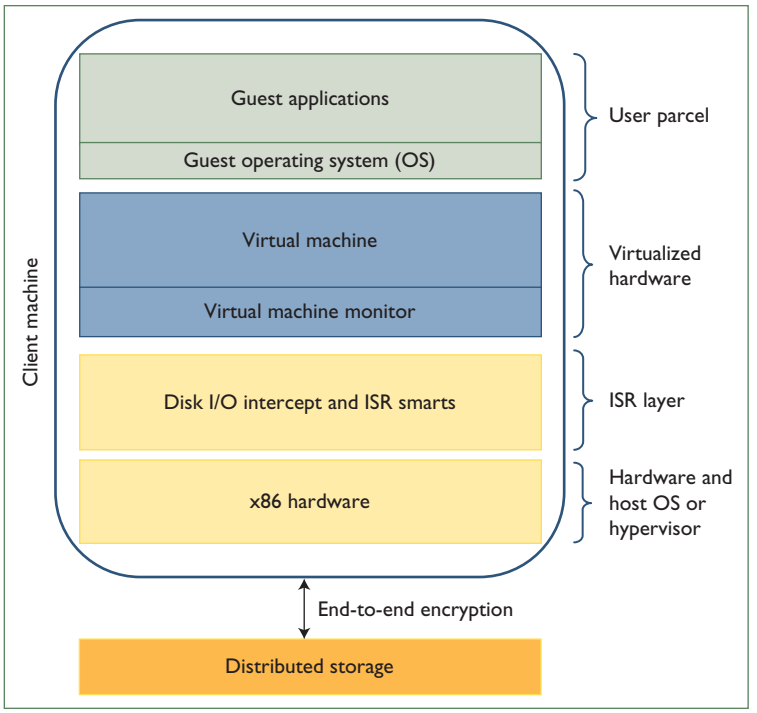

Figure 1. Modular structure of an Internet Suspend/Resume (ISR) client.

nection, the user should still be able to continue working. This challenging requirement precludes a range of thin-client solutions such as Sun Ray,[2] Virtual Network Computing (VNC),[3] and AJAX-based applications[4] that execute on a remote computing server.

- *Establish trust in unmanaged hardware for transient use.* Today, when users sit down to use a computer in their office or home, they implicitly assume that their machine hasn't been tampered with and that no malware such as a keystroke logger has been installed. This is a reasonable assumption due to restricted physical access to the machine. The same assumption applies to a portable computer that the user physically safeguards at all times. If transient use of hardware is to become commonplace, users must be able to quickly establish a similar level of confidence in hardware that they don't own or manage.

Since 2001, we've been exploring solutions to these problems in the context of ISR. As its name suggests, ISR emulates the suspend/resume capability of laptop hardware. Applications and operating systems today already support this well-understood metaphor. The difference is, of course, that ISR lets you suspend on one machine and seamlessly resume on another. We have built three experimental versions of ISR (ISR-1, ISR-2, and ISR-3) and have gained small-scale deployment experience with ISR-3. Based on our implementation and usage experience, we're working toward a new version of ISR, which we call OpenISR.

ISR Architecture

ISR builds on two technologies that have matured in the past few years. Specifically, it layers virtual machine technology on distributed storage technology. Each VM encapsulates a distinct execution and user-customization state that we call a *parcel*. The distributed storage layer transports a parcel across space (from suspend site to resume site) and time (from suspend instant to resume instant). Users can own multiple parcels, just as they can own multiple machines with different operating systems or application suites.

Figure 1 shows an ISR client machine's logical structure. This structure has remained invariant across the many different ISR versions, although the components implementing each layer have changed over time. For example, the virtual machine monitor (VMM) was VMware[5] in early versions of ISR but now can be VMware, KVM, or Xen.[6] These VMMs support a mode in which a local disk partition holds the VM state. By intercepting disk I/O references from the VMM to this disk partition, the ISR layer can transparently redirect the references to distributed storage.

As Figure 1 shows, the ISR client software encrypts data from a parcel before handing it to the distributed storage layer. Neither servers nor persistent client caches used by the distributed storage mechanism contain any unencrypted user state. Compromised storage servers can, at worst, result in a denial of service. Compromise of a client after a user suspends can, at worst, prevent the updated VM state from being propagated to servers, also resulting in denial of service. Even in these situations, ISR preserves the privacy and integrity of user parcels.

The highly asymmetric separation of concerns made possible by a distributed storage system reduces the skill level needed to manage ISR sites. Little skill is needed to maintain client machines or to deploy new ones. System administration tasks that require expertise (such as backup, restoration, load balancing, and adding new users) are concentrated on a few remotely located

servers administered by a small professional staff. We expect that server hardware and the professional staff that will administer it will often be dedicated to a specific organization such as a company, university, or ISP. Because ISR users belonging to many different organizations are likely to visit locations such as coffee shops and doctors' offices, domain-bridging mechanisms such as AFS cells[7] or Kerberos realms[8] will be valuable. Figure 2 illustrates how we might organize an ISR deployment.

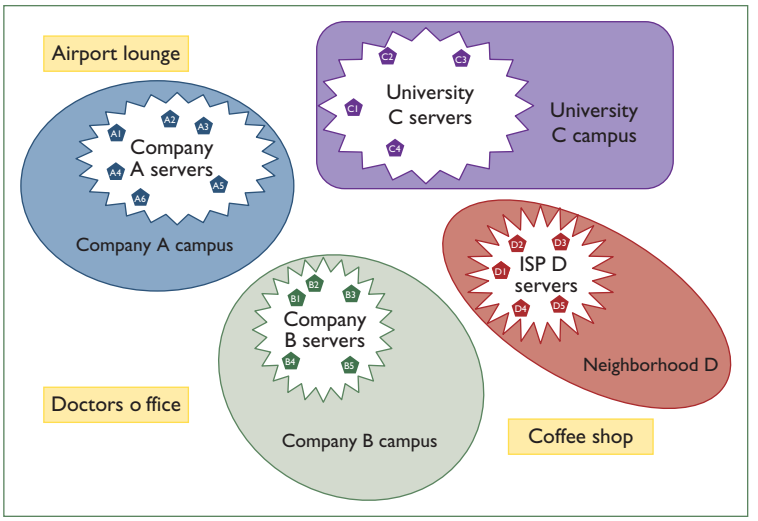

Figure 2. A hypothetical multiorganization Internet Suspend/Resume deployment.

ISR-1

ISR-1 was a proof-of-concept implementation that used NFSv3 as the distributed storage layer and VMware Workstation 3.0 on a Linux host as the VMM. The ISR layer implemented a trivial copyin/copyout of the entire VM state between file servers and the local disk. By the end of 2001, ISR-1 had confirmed that layering a VM on distributed storage could yield functionally seamless suspend and resume capability.[9] At the same time, measurements of the prototype exposed the need for a more sophisticated implementation to achieve acceptable performance.

For a user parcel configured with 128 Mbytes of main memory and a 2-Gbyte virtual disk, suspend and resume operations took roughly two minutes each on a 100 Mbits per second (Mbps) Ethernet. Although this might be tolerable for some usage models, it's much longer than modern laptops' typical suspend/resume times. Furthermore, using NFS meant that the prototype was acutely sensitive to network quality.

In spite of its shortcomings, even this simple prototype gave us useful insight into user expectations regarding suspend and resume. We realized that users perceive resume latency more acutely than suspend latency because the suspend operation can be asynchronous. Users can depart immediately after initiating suspend, for example, and let the operation complete while they're traveling. This led us to use simple file compression to shorten resume latency at the cost of increased suspend latency.

ISR-2

Encouraged by the results from ISR-1, we built a new implementation, ISR-2, and used it from early 2002 until late 2004 to explore ISR performance trade-offs.[10,11] This version of ISR had much improved performance relative to ISR-1. It also supported disconnected and weakly connected operation and enabled use of portable storage devices for accelerating VM state transfers.

In terms of the layers in Figure 1, the VMM layer continued to be the VMware Workstation 3.0 on a Linux host. However, the distributed storage layer in ISR-2 was the Coda File System[12] rather than NFS. We implemented the ISR layer in two parts. One part was a loadable kernel module called Fauxide that served as the device driver for a pseudodevice. We configured VMware to use this pseudodevice for the VM state. Fauxide redirected VMware requests to this pseudodevice to a user-level process called *Vulpes*, which was the ISR layer's second component. Vulpes implemented the VM state-transfer policy, the VM state's mapping to a directory tree of 256-Kbyte files in Coda, and hoarding control for these files.

Because Vulpes was outside the kernel and fully under our control, it was easy to experiment with a range of VM state-transfer policies. From a user's viewpoint, two questions frame ISR's key performance metrics. The first metric is *resume latency*: how soon can I begin working after resume? The second metric is *slowdown*: how sluggish is work after I resume? An ideal ISR implementation would have zero resume latency and zero slowdown. In practice, there are trade-offs between the two because policies that shrink one might increase the other.

Our results showed that a *pure demand–fetch policy* could yield resume latency as low as 14 seconds on a 100 Mbps local area network (LAN) for a modestly configured VM. Such a policy fetches only the minimal state needed for resume and

obtains the rest on demand as execution proceeds. We measured slowdown to be roughly 8 percent, at 100 Mbps. Network bandwidth was a critical factor in determining both resume latency and slowdown, rendering this policy unusable below approximately 10 Mbps.

Our results also showed that a *fully proactive policy* was usable even at dialup speeds. This policy exploits advance knowledge of the resume site to overlap VM state transfer with user travel time from the suspend to resume site. The advance knowledge might be trivially available in some situations, such as ISR being used between home and work. We measured resume latencies of approximately 10 seconds at all bandwidths, with virtually no observable slowdown at any bandwidth. Of course, this policy is only feasible when travel time exceeds VM state transfer time. In our experiments, this ranged between 45 seconds on a 100 Mbps network and 14 minutes at a DSL or cable modem speed of 1 Mbps. The latter figure hints at deployment feasibility even today because many commutes are longer than 14 minutes in major cities.

Our results also showed the considerable value of using portable storage to accelerate VM state transfers. Although our discussion of ISR until this point has emphasized its hands-free or carry-nothing aspect, we expect that users might be willing to carry something small and unobtrusive if it would enhance their ISR usage experience. Today, USB and Firewire storage devices in the form of storage key chains or microdrives are widely available. By serving as a local source of critical data, such a device could improve ISR performance at sites with poor network connectivity.

Our approach to integrating portable storage with distributed storage is called *lookaside caching*. (See an earlier work for full details.[13]) The essence of lookaside caching is to treat data on a portable storage device only as a hint, not as the authoritative version of that data. Before using the data, an ISR client compares the cryptographic hash to that provided by the server for the authoritative version. If the hashes match, the client can use a copy operation from the portable storage device instead of a fetch operation from the server over a slow network. Lookaside caching is thus an idiot-proof approach to using portable storage for ISR. If users forget to update the device at suspend, or if they absentmindedly pick up the wrong device for travel, there's no danger of using an incorrect VM state. The only consequence is a large resume latency and slowdown at sites with poor Internet connectivity. This contrasts with approaches such as SoulPad,[14] in which portable storage is assumed to contain a VM's authoritative state. Our measurements confirmed that lookaside caching can yield usable resume latencies and slowdowns at bandwidths down to 1 Mbps. (Full details of experimental results with ISR-2 are available elsewhere.[11])

ISR-3

In late 2004, we turned our attention to a real-world deployment of ISR with the goal of gaining usage experience with this new paradigm. This required us to create a new version of ISR that paid careful attention to the logistics of deployment. ISR-3[15] subsumes much of ISR-2's code and functionality but offers simpler installation and usage as well as greater flexibility in system configuration.

A major change in this version is that Coda is only one of many possible mechanisms that we can use for the distributed storage layer. ISR-3's structure makes it easy to replace Coda with alternatives, such as OpenAFS or Lustre,[16] or to use a built-in storage layer based on HTTP and Secure Socket Shell (SSH). Relative to ISR-2, the ISR layer's Fauxide component is unchanged, but we substantially modified the Vulpes component. We also upgraded the VMM to VMware Workstation 4.5.

A pilot deployment of ISR began in January 2005 and continued until mid-2006. At its peak, the ISR user population spanned 23 active users made up of Carnegie Mellon students and staff. We gave users the choice of a Windows XP parcel, a Linux parcel, or both. During the course of the pilot, users performed numerous check-in operations, eventually creating 817 distinct parcel versions. In August 2005, after seven months of continuous deployment, we took a snapshot of the memory and disk images of these parcel versions and analyzed it in detail.

A previous paper[17] reports on empirical data from this deployment, but we can summarize the highlights here. A question of particular interest to us is the effectiveness of *content addressable storage* (CAS) in reducing the storage requirements on ISR servers to retain multiple suspend images of each parcel. Such retention can be valuable, for example, in letting users easily roll back their execution states to a point before some nasty event

such as a virus infection. Because substantial similarity exists between a parcel's successive suspend images, there's potential for reducing the total storage requirements of N images to much less than ($N \times$ average parcel size). Only empirical data from real use can provide realistic estimates of these savings.

Our study yielded the surprising result that we should store the VM state on servers at much finer granularity (4 to 16 Kbytes) rather than the larger granularities (128 to 256 Kbytes) that we had used in ISR-2 and ISR-3. Our original choice of granularity was based on the trade-off between increased metadata size and an improved cache hit ratio resulting from fine granularity.[11] Including CAS in the trade-off space produces a very different result: the benefit from improved similarity across parcel images outweighs the metadata overhead of finer granularity.

At a 4-Kbyte granularity, our results show that using CAS could reduce server storage requirements by approximately 60 percent relative to a simple block-based differencing policy. The corresponding network bandwidth savings is 70 percent. Our results also quantify the trade-off between privacy and exploiting similarity across user parcels. If the encryption-key management policy allows servers to detect similarity across user parcels, the storage and network bandwidth savings are higher — 80 percent each, relative to a block-based similarity-detection policy.

The OpenISR Version

Based on our experience and insights from previous versions of ISR and from our related work on transparently morphing between thick- and thin-client modes of execution,[18] we've begun implementing the OpenISR version of ISR. In addition to extensive use of CAS, several key ideas distinguish this version from its predecessors.

VMM Agnosticism

The past few years have seen an explosion of VMMs for the Intel x86 hardware. Examples include VMware Workstation, VMware ESX, Xen, KVM, Microsoft Virtual Server, and Parallels Workstation. To give users the greatest freedom in choice of resume site, ISR should place the fewest possible constraints on the site configuration. In particular, it should let resume occur on any x86 machine with a VMM, even if that VMM is different from the one at the most recent suspend site. We refer to this property as *VMM agnosticism*. To achieve this, we store a parcel's virtual disk components in a VMM-independent format on servers. Only the memory image is in a VMM-specific format. If a mismatch occurs between the VMM types of the suspend and resume sites, we treat this as an ISR exception.

We're exploring several approaches to handling this exception. For example, one approach is to transform the memory image's format. Another is to convert the resume operation into a reboot operation, after user warning and approval.

Transient Thin-Client Mode

The time to transfer a suspended VM's memory image is a lower bound on resume latency in ISR-2 and ISR-3. Even with a pure demand–fetch policy for the disk image, the state corresponding to a VM's entire physical memory must be available at the resume site before execution begins. This limits the usefulness of ISR for transient use at locations such as a coffee shop or doctor's office, where a user might only have a few minutes to spare. To overcome this limitation, we're building on techniques that we've developed to transparently morph between thin- and thick-client execution modes.

We envision a usage model in which ISR is responsible for instantiating a user's customized operating system and applications on demand.

Our work to date has confirmed the feasibility of using VMs as the basis of morphing between thin-client execution during the resource-intensive computational phases of an application and thick-client execution during its interaction-intensive phases. We plan to exploit this capability to achieve the lowest possible resume latency. Upon resume, execution begins in thin-client mode. During an initial brief period of user interaction, sufficient VM state is transferred in the background to switch to thick-client mode with a demand–fetch policy. For some extremely brief sessions, the user might suspend before execution switches to thick-client mode.

Guest-Aware Migration

In all previous ISR versions, we treated the guest operating system as a black box. This has the attractive property that no modifications are needed to the guest or to applications running on it. ISR can therefore be used with proprietary operating systems such as Windows XP and proprietary applications such as the Microsoft Office suite. However, this black-box approach also implies that ISR can't exploit knowledge the guest operating system possesses.

Consider the guest operating system's virtual memory manager shortly before a suspend operation. If virtual memory is working well, the current working set would be in physical memory and an internal data structure approximating least recently used (LRU) would be tracking the working set. That data structure contains precious knowledge that ISR could use for prefetching a VM state at resume. ISR could lower resume latency by using a demand–fetch policy for VM state, augmented with asynchronous background prefetching. Alternatively, ISR could fetch the entire working set before allowing user interaction. Except in pathological situations, prefetching based on accurate knowledge of the working set will typically yield much smaller slowdown than a pure demand–fetch policy.

The virtual disk doesn't have to … hold all of a user's files; it only has to be large enough for a Coda cache that can hold the working set of those files.

We refer to ISR implemented with help from the guest operating system as a *guest-aware implementation*. The virtual memory example we just gave is only one of many guest-aware mechanisms that we're exploring. The concept of *paravirtualization* (requiring a small amount of low-level modifications to guest operating systems) in VMMs like Xen anticipates the notion of seeking guest assistance rather than requiring total transparency.

Cross-Parcel Data Sharing

Until now, our focus in ISR has been to recreate the PC user experience, including extensive use of the local disk to store user files. A consequence of this tight emulation of the PC usage model is the absence of support for data sharing across users and even by a single user across parcels. As we move to OpenISR, we're exploring usage models that simplify data sharing. Our approach is to use Coda in the guest operating system as the data-sharing mechanism. This is in contrast to how we used Coda in ISR-2, where it was part of the host operating system and therefore invisible to the user and guest applications.

We envision a usage model in which ISR is responsible for instantiating a user's customized operating system and applications on demand, but Coda is responsible for providing a single, system-wide image of user data. This usage model hearkens back to the Andrew model of distributed personal computing explored nearly 20 years ago, which combines the flexibility and visually rich user-machine interface of personal computing with timesharing's ease of communication and information sharing.[19] The OpenISR approach preserves Andrew's location transparency and ease of data sharing but improves the fidelity with which a user's environment is recreated at any usage site.

This approach has some implications for ISR implementation. First, it's likely that VMs will be smaller, especially with respect to virtual disk size. The virtual disk doesn't have to be large enough to hold all of a user's files; rather it only has to be large enough for a Coda cache that can hold the working set of those files. Second, there are opportunities to make Coda ISR-aware and vice versa. For example, as with virtual memory, ISR can infer prefetching hints from the LRU information on cached files that Coda maintains in the guest operating system.

Resilience to Internet Vagaries

So far in this article, we've mainly focused on providing on-demand access to a user's personal computing environment. We now turn to insulating users from the Internet's vagaries. Once data is fully hoarded, ISR doesn't require the network to be available. The underlying storage system's disconnected operation capability provides the illusion of connectivity for ISR. Users can use the cached state even when disconnected. The client buffers updates and eventually reintegrates them when network connectivity is restored. There's no danger of conflicting updates on reintegration

because ISR enforces a single-writer model at VM granularity — resume occurs only after a lock on the entire VM state is acquired from ISR servers.

Thus, ISR is *asynchronous* in its network dependence. Connectivity is necessary while hoarding data and during eventual reintegration. For extended periods between these two events (possibly lasting many hours), total disconnection is acceptable and has absolutely no performance impact. If the ISR client is a laptop, a user can be as mobile and productive with it during the disconnection period as he or she is with a laptop today. Of course, the setup won't support direct use of the network (such as for Web browsing) while disconnected. But users can perform all other work (such as editing, authoring, and so on) that doesn't require network access.

ISR's asynchronous network dependence distinguishes it from the *synchronous* network dependence of thin clients. By definition, disconnected operation is impossible with thin clients. Furthermore, network quality must be sufficient at all times for crisp, interactive response. (Note that it's the worst case, not the average case, that determines whether a thin-client approach will be satisfactory.[1]) In addition to physical-layer transmission delays and end-to-end software path lengths, technologies such as firewalls, overlay networks, and lossy wireless networks add latency and other hurdles. Even with a pure demand–fetch policy, network latency affects ISR performance much less than thin-client performance. This is especially true for intensely interactive applications.

Interest in thin clients is high today because of frustration with PCs' high ownership costs. Unfortunately, dependence on thin clients might hurt the important goal of crisp, interactive response. Furthermore, an ISR client can leverage local graphics hardware accelerators, an increasingly common feature of today's computing landscape. Extensive evidence exists from the human–computer interaction community that interactive response times over 150 milliseconds are noticeable and begin to annoy a user as they approach one second. To achieve seamless mobility with thin clients, we need tight control of end-to-end network latency, which is difficult at Internet scale. Adding bandwidth is relatively easy, but reducing latency is much harder. We see ISR as a solution that trades off startup delay for crisp interaction. Once execution begins, all interaction is local. At the same time, an ISR client matches a thin client's valuable property: it doesn't contain any long-term user state.

Establishing Trust

The third problem we described in the "Liberating Personal Computing" section is establishing trust in unmanaged machines for transient use. To address this problem, we're creating a tool called Trust-Sniffer that helps a user incrementally gain confidence in a machine that is initially untrusted. Trust-Sniffer focuses on software attacks but doesn't guard against hardware attacks. Only physical surveillance or the use of tamper-proof or tamper-evident hardware could guard against hardware attacks such as modifying the basic input/output system (BIOS).

The root of trust in Trust-Sniffer is a small, lightweight device such as a USB storage device that users own and carry with them at all times. This trust initiation device boots the untrusted machine so that Trust-Sniffer can examine its local disk and verify the integrity of all software that would be used in a normal boot process. The integrity check compares the SHA-1 hashes of the kernel image and related boot software on disk against a list located on the trust initiation device. The list has the hashes of software that's known to be good. After verifying the normal boot process's integrity, Trust-Sniffer performs a reboot from disk.

As ISR gains momentum, we envision a process of interleaved development and deployment in which each drives the other.

In this step, Trust-Sniffer's *trust-extender module* is dynamically loaded into the kernel. As its name implies, this module is responsible for extending the zone of trust as execution proceeds. On the first attempt to execute any code that lies outside the current zone of trust (including dynamically linked libraries), the kernel triggers a *trust fault*. To handle a trust fault, the trust extender verifies the suspect module's integrity by comparing its SHA-1 hash against that of known, good

modules. Execution stops if the trust extender can't establish the suspect module's integrity.

This staged approach to establishing confidence in an untrusted machine strikes a good balance between the needs of security, usability, and speed. Once users have gained confidence in a machine and its ISR software, they can perform ISR's resume step.

We plan to make the first OpenISR software release in early 2007. The implementation involves a complete replacement of the Fauxide module shown in Figure 1 by a new loadable kernel module called Nexus. Our usage experience with ISR-3 has convinced us that a complete restructuring and replacement of this component is necessary to avoid subtle deadlocking issues that arise under conditions of heavy memory pressure. To conform to this restructuring, the Vulpes module in Figure 1 will also undergo major changes that will result in a new module. Although these are extensive client changes, our plan is to initially preserve the server code with minimal changes. This will let us begin deployment of OpenISR with limited functionality to the development team to gain hands-on usage experience as soon as possible. We'll then implement new server code that will use established authentication mechanisms such as Kerberos and Active Directory Services, avoiding the need for users to have SSH privileges on servers (an ISR-3 requirement). We see this as a critical requirement for expanding usage beyond a small deployment.

As ISR gains momentum, we envision a process of interleaved development and deployment in which each drives the other. The scale of eventual deployment will depend on the funding resources that we can obtain. A campus-scale deployment is our dream. Such a deployment, on the order of thousands of users, would be a strong validation of the ISR vision. Once it reaches that scale, we're confident that this tantalizing vision will become self-sustaining.

Acknowledgments

Internet Suspend/Resume is a registered trademark and OpenISR is a trademark of Carnegie Mellon University. All unidentified trademarks mentioned in this article are properties of their respective owners. This research was supported by the US National Science Foundation (NSF) under grant numbers CNS-0509004 and CCR-0205266, the National Science and Engineering Research Council (NSERC) of Canada under grant number 261545-3 and a Canada Graduate Scholarship, the Canadian Foundation for Innovation (CFI), and the Ontario Innovation Trust (OIT) under grant number 7739. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and don't necessarily reflect the views of the NSF, NSERC, CFI, OIT, Carnegie Mellon University, the University of Toronto, Intel, or Pennsylvania State University.

References

1. N. Tolia, D.G. Andersen, and M. Satyanarayanan, "Quantifying Interactive User Experience on Thin Clients," *Computer*, vol. 39, no. 3, 2006, pp. 46–52.
2. P. Venezia, "Sun Whips Its Sun Ray Thin Client into Better Shape," *InfoWorld*, Aug. 2006.
3. T. Richardson et al., "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, 1998, pp. 33–38.
4. D. Johnson, "Ajax: Dawn of a New Developer," *Java World*, Oct. 2005.
5. L. Grinzo, "Getting Virtual with VMware 2.0," *Linux Magazine*, June 2000.
6. P. Barham et al., "Xen and the Art of Virtualization," *Proc. 19th ACM Symp. Operating Systems Principles*, ACM Press, 2003, pp. 164–177.
7. R. Campbell, *Managing AFS: The Andrew File System*, Prentice Hall, 1998.
8. J.G. Steiner, C. Neuman, and J.I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," *Usenix Conf. Proc.*, Usenix Assoc., 1988, pp. 191–202.
9. M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," *Proc. 4th IEEE Workshop Mobile Computing Systems and Applications*, IEEE CS Press, 2002, p. 40.
10. M. Kozuch et al., "Seamless Mobile Computing on Fixed Infrastructure," *Computer*, vol. 37, no. 7, 2004, pp. 65–72.
11. M. Satyanaranyanan et al., "Towards Seamless Mobility on Pervasive Hardware," *Pervasive and Mobile Computing*, vol. 1, no. 2, 2005, pp. 157–189.
12. M. Satyanaranyanan, "The Evolution of Coda," *ACM Trans. Computer Systems*, vol. 20, no. 2, 2002, pp. 85–124.
13. N. Tolia et al., "Integrating Portable and Distributed Storage," *Proc. 3rd Usenix Conf. File and Storage Technologies*, Usenix Assoc., 2004.
14. R. Caceres et al., "Reincarnating PCs with Portable SoulPads," *Proc. Mobisys 2005: 3rd Int'l Conf. Mobile Systems, Applications, and Services*, Usenix Assoc., 2005.
15. M. Kozuch et al., "Enterprise Client Management with Internet Suspend/ Resume," *Intel Technical J.*, vol. 8, no. 4, 2004.
16. P. Schwann, "Lustre: Building a File System for 1,000-node Clusters," *Proc. 2003 Linux Symp.*, 2003; http://archive.linuxsymposium.org/ols2003/Proceedings/All-Reprints/Reprint-Schwan-OLS2003.pdf.
17. P. Nath et al., "Design Tradeoffs in Applying Content

Addressable Storage to Enterprise-Scale Systems Based on Virtual Machines," *Proc. Usenix Technical Conf.*, Usenix Assoc., 2006.
18. A. Lagar-Cavilla et al., *Dimorphic Computing*, tech. report CMU-CS-06-123, Computer Science Dept., Carnegie Mellon Univ., 2006.
19. J.H. Morris et al., "Andrew: A Distributed Personal Computing Environment," *Comm. ACM*, vol. 29, no. 3, 1986, pp. 184–201.

Mahadev Satyanarayanan is the Carnegie Group Professor of Computer Science at Carnegie Mellon University. His research interests include mobile computing, pervasive computing, and distributed systems. Satyanarayanan has a PhD in computer science from Carnegie Mellon University. He's a fellow of the ACM and the IEEE. Contact him at satya@cs.cmu.edu.

Benjamin Gilbert is a research scholar at Carnegie Mellon University. His research interests include networking and virtualization. Gilbert has an MS in electrical and computer engineering from Carnegie Mellon University. Contact him at bgilbert@cs.cmu.edu.

Matt Toups is a research programmer at Carnegie Mellon University. His research interests include operating systems and networking. Toups is working toward his BS in computer science at Carnegie Mellon University. Contact him at mtoups@cs.cmu.edu.

Niraj Tolia is a graduate student in electrical and computer engineering at Carnegie Mellon University. His research interests include content-addressable storage systems and virtualization. Tolia has an MS in electrical and computer engineering from Carnegie Mellon University. He's a member of the ACM and Usenix. Contact him at ntolia@cmu.edu.

David R. O'Hallaron is an associate professor of computer science and electrical and computer engineering at Carnegie Mellon University. His research interests include mobile computing, computational database systems, and large-scale scientific computing. O'Hallaron has a PhD in computer science from the University of Virginia. He's a member of ACM and IEEE. Contact him at droh@cs.cmu.edu.

Ajay Surie is a graduate student in computer science at Carnegie Mellon University. His research interests include distributed systems and security. Surie has a BS in computer science from Carnegie Mellon University. Contact him at asurie@cs.cmu.edu.

Adam Wolbach is a graduate student in computer science at Carnegie Mellon University. His research interests include distributed systems. Wolbach has a BS in computer science from Carnegie Mellon University. Contact him at awolbach@cs.cmu.edu.

Jan Harkes is a senior project scientist at Carnegie Mellon University. His research interests include distributed systems, networking, and operating systems. Harkes has an MS in computer science from the Vrije University, Amsterdam. Contact him at jaharkes@cs.cmu.edu.

Adrian Perrig is an assistant professor at Carnegie Mellon University. His research interests include networking and systems security and security for mobile computing and sensor networks. Perrig has a PhD in computer science from Carnegie Mellon University. He is a member of the ACM and the IEEE. Contact him at adrian@ece.cmu.edu.

David J. Farber is a Distinguished Career Professor of Computer Science and Public Policy at Carnegie Mellon University. His research interests include networking, distributed systems, and public policy. Farber has an honorary degree of Doctor of Engineering from the Stevens Institute of Technology. He is a fellow of the ACM and the IEEE. Contact him at dfarber@cs.cmu.edu.

Michael A. Kozuch is a principal researcher at Intel Research Pittsburgh. His research interests include novel uses of virtual machine technology. Kozuch has a PhD in electrical engineering from Princeton University. Contact him at michael.a.kozuch@intel.com.

Casey J. Helfrich is a research engineer at Intel Research Pittsburgh. His research interests include virtualization, data distribution, and large-scale simulation. Helfrich has a BS in computer science and physics from Carnegie Mellon University. Contact him at casey.j.helfrich@intel.com.

Partho Nath is a graduate student in computer science and engineering at Pennsylvania State University. His research interests include high-performance storage systems, file systems, and virtual machines. Nath has a BS in computer science from IT-BHU, India. Contact him at nath@cse.psu.edu.

H. Andrés Lagar-Cavilla is a graduate student in computer science at the University of Toronto. His research interests include mobile computing, distributed systems, and virtualization. Lagar-Cavilla has an MSc in computer science from the University of Toronto. He currently holds an NSERC Doctoral Canada Graduate Scholarship. Contact him at andreslc@cs.toronto.edu.