To Carry or To Find?

Footloose on the Internet with a zero-pound laptop

M. Satyanarayanan, Benjamin Gilbert, Niraj Tolia,
H. Andrés Lagar-Cavilla[†], Ajay Surie, Partho Nath[•], Adam Wolbach,
Jan Harkes, Matt Toups, Michael A. Kozuch[‡], Casey J. Helfrich[‡],
David R. O'Hallaron, Adrian Perrig, David J. Farber

September 2006
CMU-CS-06-158

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Carnegie Mellon University, [†]University of Toronto, [‡]Intel Research Pittsburgh, [•]Pennsylvania State University

Abstract

Internet Suspend/Resume (ISR) is a new model of personal computing that cuts the tight binding between personal computing state and personal computing hardware. ISR is implemented by layering a virtual machine (VM) on distributed storage. The VM encapsulates execution and user customization state; distributed storage transports that state across space and time. In this paper, we explore the implications of ISR for an infrastructure-based approach to mobile computing. We report on our experience with three versions of ISR, and describe work in progress on a new version called *OpenISR.*

This research was supported by the 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, by 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 do not necessarily reflect the views of the NSF, NSERC, CFI, OIT, Carnegie Mellon University, the University of Toronto, Intel Corporation, or Pennsylvania State University.

Keywords: mobile computing, virtual machine, Internet Suspend/Resume, ISR, OpenISR, operating system, Coda, content-addressable storage, CAS, VMware, Xen

1 Introduction

Portable computers have been the driving technology of mobile computing since the early 1990s. Today, the term "mobile computing" is almost synonymous with the use of laptop or handheld computers. However, the plummeting cost of hardware suggests that pervasive computing infrastructure may some day eliminate the need to carry such hardware. In this paper, we describe a new approach to mobile computing that embraces this opportunity. Rather than carrying hardware, one finds and uses hardware transiently at any location. Imagine a world where coffee shops, airport lounges, dental and medical offices, and other semi-public spaces provide hardware for their clientele. Even the fold-out tray at every seat in an aircraft 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.

Why is infrastructure-based mobile computing attractive? There are obvious advantages such as travelling with less luggage, simplifying security screening in a post-9/11 world, and being able to get work done at unexpected times and locations even if one didn't have the foresight to bring one's 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. While mobile elements will improve in absolute ability, they will always be resource-poor relative to static elements of the same vintage. Further, the dependence on a finite energy source and the need to monitor remaining energy is an ongoing distraction for a mobile user.

2 Liberating Personal Computing

The world we envision retains the user customization aspect of personal computing. However, computers themselves become a ubiquitous resource, much like light at the flip of a switch, water from a faucet, or the air we breathe. In other words, personal computing is liberated from its tight binding to hardware. On demand, any Internet-connected computer can temporarily become one's personal computer. When a user starts to use the machine, it acquires his unique customization and state from a server. When he is done, his modified state is erased from that machine and returned to the server. Loss, theft or destruction of the machine is only a minor incovenience, not a catastrophic event for the user.

To attain this vision, one needs to solve at least three difficult technical problems:

- *Providing efficient on-demand access to a user's* entire *personal computing environment.*
  Today, a user who carries a portable computer is assured of seeing exactly the same set of personal files, operating system, customizations, and so on everywhere he goes. Precise and complete recreation of this familiar context is the key to low user distraction in any future model of mobile computing. Just providing access to a user's personal files or to application customizations will not suffice.

- *Ensuring resilience to Internet vagaries*
  Today, a user working with local data on his portable computer is unaffected by network quality. He is not impacted by unpredictable bandwidth and latency, or by occasional failures. This defines the standard against which new models of mobile computing will be judged. The user must perceive crisp and 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

pop-up menus all assume a very tight feedback loop between the user and his application. Only a "thick client" solution, in which the application executes very close to the user, can support this tight feedback loop when network connectivity is poor [17]. In the extreme case of network disconnection, the user should be still be able to continue work. This challenging requirement precludes a wide range of "thin client" solutions such as Sun Ray [19], VNC [12] and AJAX-based applications [5], in which applications execute on a remote compute server.

- *Establishing trust in unmanaged hardware for transient use.*
  Today, when a user sits down to use a computer in his office or home, he implicitly assumes that the machine has not been tampered with and that no malware such as a keystroke logger has been installed. This is a reasonable assumption today because physical access to the machine is restricted. The same assumption applies to a portable computer that is physically safeguarded by the user at all times. If transient use of hardware is to become commonplace, it is necessary for a user to be able to quickly establish a similar level of confidence in hardware that he does not own or manage.

3 Internet Suspend/Resume

Since 2001, we have been exploring solutions to these problems in the context of a system called *Internet Suspend/Resume*^{TM}*(ISR)*. 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 you to suspend on one machine and seamlessly resume on another. We have built three experimental versions of ISR, called 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 are working towards a new version of ISR called OpenISR^{TM}.

3.1 Architecture

ISR builds on two technologies that have matured in the past few years. Specifically, it layers *virtual machine* (VM) 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). It is possible for a user to own multiple parcels, just as he can own multiple machines with different operating systems or application suites today.

Figure 1 shows the logical structure of an ISR client machine. This logical structure has remained invariant across the many different versions of ISR, although the components implementing each layer have changed over time. For example, the Virtual Machine Monitor (VMM) was VMware [4] in early versions of ISR, but can be VMware or Xen [1] in OpenISR. Both VMMs support a mode in which a local disk partition holds 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 shown in Figure 1, data from a parcel is encrypted by ISR client software before it is handed to the distributed storage layer. Neither servers nor persistent client caches used by the distributed storage mechanism contain any unencrypted user state. Compromise of storage servers can, at worst, result in denial of service. Compromise of a client after a user suspends can at worst prevent updated VM state from being propagated to servers, also resulting in denial of service. Even in these situations, the privacy and integrity of user parcels is preserved.

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 machines or to deploy new ones.

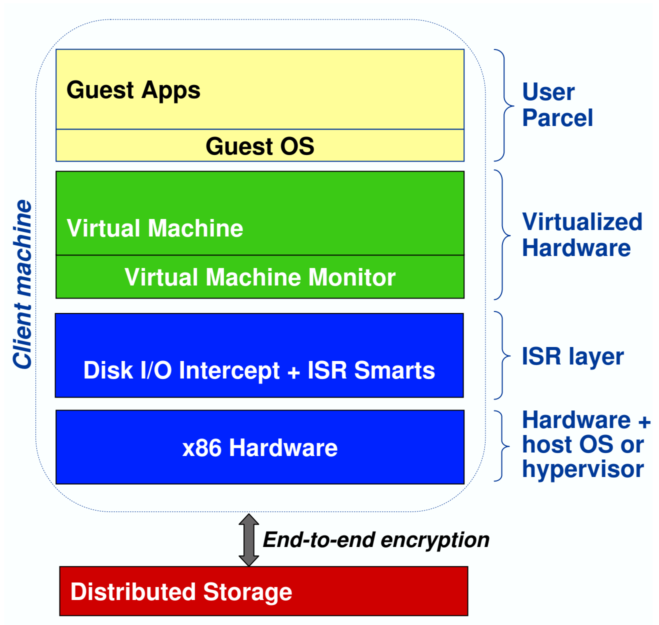

Figure 1: Logical ISR Client Structure

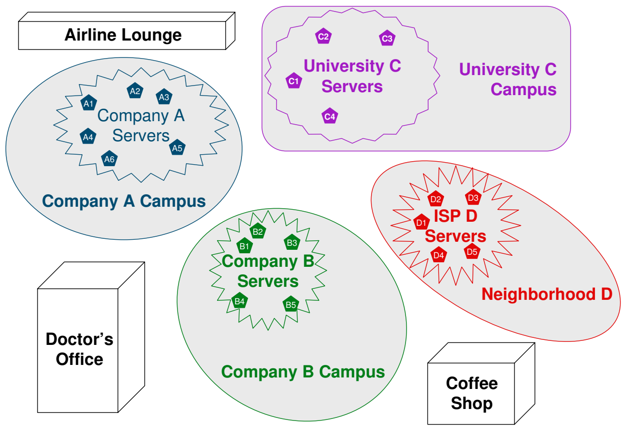

Figure 2: Hypothetical ISR Deployment

System administration tasks that require expertise (such as backup, restoration, load balancing, and addition of 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 to administer them will often be dedicated to a specific organization such as a company, university or ISP. Since locations such as coffee shops and doctors' offices are likely to be visited by ISR users belonging to many different organizations, domain-bridging mechanisms such as AFS *cells* [3] or Kerberos *realms* [16] will be valuable. Figure 2 illustrates how a deployment of ISR might be organized.

3.2 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 entire VM state between NFS and files on the local disk. By the end of 2001, ISR-1 had confirmed that layering a VM on distributed storage could indeed yield functionally seamless suspend and resume capability [7]. 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 MB of main memory and a 2 GB virtual disk, suspend and resume operations took roughly two minutes each on a 100 Mb/s Ethernet. While this might be tolerable for some usage models, it is much longer than the typical suspend/resume times of modern laptops. Further, the use of 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. For example, a user can depart immediately after initiating suspend and allow the operation to complete while he is travelling. This led to the use of simple file compression to shorten resume latency at the cost of increased suspend latency.

3.3 ISR-2

Encouraged by the results from ISR-1, we built a completely new implementation called ISR-2 and used it from early 2002 until late 2004 to explore ISR performance tradeoffs [8, 13]. 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 of Figure 1, the VMM layer continued to be VMware Workstation 3.0 on a Linux host. However, the distributed storage layer in ISR-2 was the Coda File System [14] rather than NFS. The ISR layer was implemented in two parts. One part was a loadable kernel module called *Fauxide* that served as the device driver for a pseudo-device. VMware was configured to use this pseudo-device for VM state. Disk I/O requests to this pseudo-device by VMware were redirected by Fauxide to a user-level process called *Vulpes,* that constituted the second component of the ISR layer. Vulpes implemented the VM state transfer policy, the mapping of VM state to a directory tree of 256KB files in Coda, and hoarding control for these files.

Since Vulpes was outside the kernel and fully under our control, it was easy to experiment with a wide range of VM state transfer policies. From a user's viewpoint, two questions frame the key performance metrics of ISR. The first metric is called *resume latency,* and answers the question "How soon can I begin working after resume?" The second metric is called *slowdown,* and answers the question "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: policies that shrink one may increase the other.

Our results showed that a *pure demand-fetch* policy could yield resume latency as low as 14 seconds on a 100 Mb/s 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. Slowdown was measured to be roughly 8% at 100Mb/s. Network bandwidth was observed to be a critical factor in determining both resume latency and slowdown, rendering this policy unusable below about 10Mb/s. Our results also showed that a *fully proactive* policy was usable even at a dialup speeds. This policy exploits advance knowledge of resume site to overlap VM state transfer with user travel time from suspend to resume site. The advance knowledge may be trivially available in some situations, such as ISR being used between home and work. Resume latencies of approximately 10 seconds were measured at all bandwidths, with virtually no observable slowdown at any bandwidth. Of course, a fully proactive policy is only feasible when travel time exceeds VM state transfer time. In our experiments, this ranged between 45 seconds on a 100 Mb/s network and 14 minutes at a DSL or cable modem speed of 1 Mb/s. The latter figure hints at deployment feasibility even today, since 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 emphasised its "hands-free" or "carry-nothing" aspect, we expect that users may be willing to carry something small and unobtrusive if that would enhance their ISR usage experience. Today, USB and Firewire storage devices in the form of storage keychains 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,* and is fully described in an earlier paper [18]. 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, its cryptographic hash is compared to that provided by the server for the authoritative version. If the hashes match, a copy operation from the portable storage device can be used instead of a fetch operation from the server over a slow network. Lookaside caching is thus an "idiot-proof" approach to the use of portable storage for ISR. If a user forgets to update the device at suspend, or if he absent-mindedly picks up the wrong device for travel, there is no danger of using incorrect VM state. The only consequence is large resume latency and slowdown at sites with poor Internet connectivity. This is in contrast to approaches such as SoulPad [2] in which portable storage is assumed to contain the authoritative state of a VM. Our measurements confirmed that lookaside caching can yield very usable resume latencies and slowdowns at bandwidths down to 1 Mb/s. Full details of experimental results with ISR-2 can be found in an earlier paper [13].

3.4 ISR-3

In late 2004, we turned our attention to real-life 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, described by Kozuch et al [6], subsumes much of the code and functionality of ISR-2, but offers simpler installation and usage as well as greater flexibility in system configuration. A major change is in the distributed storage layer. In ISR-3, Coda is only one of many possible mechanisms that can be used for this layer. The structure of ISR-3 makes it easy to replace Coda with alternatives such as AFS or Lustre [15], or to use a built-in storage layer based on HTTP and SSH. Relative to ISR-2, the Fauxide component of the ISR layer is unchanged but the Vulpes component has been substantially modified. The VMM layer has been upgraded to VMware Workstation 4.5.

A pilot deployment of ISR was initiated in January 2005, and continues to the present time. At its peak, the ISR user population spanned 23 active users who were drawn from the ranks of Carnegie Mellon students and staff. Users were given the choice of a Windows XP parcel, a Linux parcel, or both. During the course of the pilot, users performed numerous checkin operations, eventually creating 817 distinct parcel versions. In August, 2005, after 7 months of continuous deployment, a snapshot of the memory and disk images of these parcel versions was taken and analyzed in detail.

A recent paper [11] reports on empirical data from this deployment. We summarize the highlights of that paper 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 allowing users to easily rollback their execution states to a point before some nasty event such as a virus infection. Since there is substantial similarity between successive suspend images of a parcel, there is potential for reducing the total storage requirements of N images to much less than $N * parcelsize$. Only empirical data from real use can provide realistic estimates of these savings.

This study yielded the surprising result that VM state should be stored on servers at much finer granularity (4–16KB) rather than the larger granularities (128–256KB) that we had used in ISR-2 and ISR-3. Our original choice of granularity was based on the trade-off between increased meta-data size and improved cache hit ratio resulting from fine granularity [13]. Including CAS in the trade-off space produces a very different result: the benefit from improved similarity across parcel images outweighs the meta-data overhead of finer granularity.

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

3.5 OpenISR

Based on our experience and insights from previous versions of ISR and from our related work on *dimorphic computing* [9], we have begun implementation of a completely new version of ISR called OpenISR. In addition to extensive use of CAS, here are the key ideas in OpenISR that distinguish it 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, Microsoft Virtual Server and Parallels Workstation. To give a user the greatest freedom in choice of resume site, ISR should place the fewest possible constraints on the site configuration. In particular, it should allow resume to 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 it, OpenISR stores the virtual disk components of a parcel in a VMM-independent format on servers. Only the memory image is in a VMM-specific format. If there is a mismatch between the VMM types of the suspend and resume sites, OpenISR treats this as an ISR exception. We are exploring a number of different approaches to handling this exception. For example, one approach is to transform the format of the memory image. Another approach is to convert the resume operation into a reboot operation, after user warning and approval.

- *Transient thin client mode*
  The time to transfer the memory image of a suspended VM 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 the entire physical memory of a VM 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 a doctor's office, where a user may only have a few minutes to spare. To overcome this limitation, we are building on techniques that we have developed in the context of dimorphic computing to transparently morph between thin and thick client modes of execution.

  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. In OpenISR, we are exploiting this capability for a different purpose: namely, to achieve the lowest possible resume latency. Upon resume, OpenISR begins execution in thin client mode. During an initial brief period of user interaction, it transfers sufficient VM state in the background to switch to thick client mode with a demand-fetch policy. For some extremely brief sessions, the user may suspend before execution switches to thick client mode.

- *Guest-aware migration*
  In all previous versions of ISR, the guest operating system has been treated 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 cannot exploit knowledge possessed by the guest operating system.

  For example, consider the virtual memory manager of the guest operating system 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 LRU would be tracking the working set. That data structure contains precious knowledge that ISR could use for prefetching 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.

  We refer to ISR implemented with help from the guest operating system as a "guest-aware" implementation. The virtual memory example given above is only one of many guest-aware mechanisms that we are exploring in OpenISR. The concept of *paravirtualization* in VMMs like Xen anticipates the notion of seeking guest assistance.

- *Cross-parcel data sharing*
  Until now, our focus in ISR has been to recreate the PC user experience. This includes 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 his parcels. As we move to OpenISR, we are 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 the use of Coda in ISR-2, where it was part of the host operating system and therefore invisible to the user and to guest applications.

  We thus 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 harkens back to the Andrew model of distributed personal computing explored nearly 20 years ago [10]. It preserves the location transparency and ease of data sharing of the Andrew model, but improves the fidelity with which a user's environment is re-created at any usage site.

  This usage model has some implications for OpenISR implementation. First, it is likely that VMs will be smaller, especially with respect to virtual disk size. The virtual disk does not 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, the LRU information on cached files maintained by Coda in the guest operating system can be exploited by ISR for prefetching.

4 Resilience to Internet Vagaries

Section 3 has focused mainly on the first problem discussed in Section 2: providing on-demand access to a user's personal computing environment. We now turn to the second problem discussed in Section 2: insulating users from the vagaries of the Internet. Once data is fully hoarded, ISR does not require the network to be available. The disconnected operation capability of the underlying storage system provides the illusion of connectivity for ISR. Cached state can be used even when disconnected. Updates are buffered by the client, and eventually reintegrated when network connectivity is restored. There is 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 needed 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 machine is a laptop, a user can be as mobile and productive with it during the period of disconnection as he is with a laptop today. Of course, direct use of the network by the user (such as Web browsing) cannot be supported while disconnected. But all other work such as editing, authoring, and so on, can be performed just as if the network were up.

The asynchronous network dependence of ISR distinguishes it from the *synchronous* network dependence of thin clients. By definition, disconnected operation is impossible with thin clients. Further, the quality of the network has to be sufficient at all times for crisp interactive response. Note that it is the worst case, not the average case, that determines whether a thin client approach will be satisfactory [17]. 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 it affects thin client performance. This is especially true for intensely interactive applications.

Interest in thin clients is very high today because of frustration with the high total cost of ownership of PCs. Unfortunately, dependence on thin clients may hurt the important goal of crisp interactive response. Further, an ISR client can leverage local graphics hardware accelerators, an increasingly common feature of today's computing landscape. There is extensive evidence from the HCI community that interactive response times over 150ms are noticeable and begin to annoy a user as they approach 1s. To attain the goal of seamless mobility with thin clients, one needs very tight control of end-to-end network latency. This is hard to achieve at Internet scale. Adding bandwidth is relatively easy, but reducing latency is much harder. ISR can be viewed 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 valuable property of a thin client: it is stateless from the viewpoint of long-term user state.

5 Establishing Trust

We now turn to the third problem described in Section 2, that of establishing trust in unmanaged machines for transient use. To address this problem, we are 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 does not 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 BIOS.

The root of trust in Trust-Sniffer is a small, lightweight device such as a USB storage device that is owned by the user and carried by him at all times. This trust initiation device is used to boot 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 is performed by comparing the SHA-1 hashes of the kernel image and related boot software on disk against a list that is located on the trust initiation device. The list has the hashes of known good software. Once the integrity of the normal boot process is verified, a reboot from disk is performed. In this step, the *trust extender* module of Trust-Sniffer 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 integrity of the suspect module by comparing its SHA-1 hash against that of known good modules. Execution stops if the module's integrity cannot be established. This staged approach to establishing confidence in an untrusted machine strikes a good balance between the needs of security, usability and speed. Once the user has gained confidence in the machine and its ISR software, he can perform the resume step of ISR.

6 Think Wallet, not Swiss Army Knife

ISR offers a fundamentally different way of thinking about mobile computing, and about the resources that one needs to carry. The current design philosophy for mobile devices can be characterized as the "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 of that functionality. For example, one 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 other functions of that device.

Contrast this with the very different design philosophy of a wallet. Virtually every adult carries a wallet. The contents vary, but typically include things like cash, credit cards, ID cards, and so on. Far from civilization, a wallet is useless. You could 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 becomes useful. With cash or a credit card, one can obtain anything one needs, when and where one needs it.

A wallet can thus be viewed as a device that helps transform generic infrastructure into highly personalized services. This observation leads to a very 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 lookaside caching brings us close to the model of carrying a wallet rather than a Swiss Army Knife. Such a device could even have the form factor of a credit card and fit into your wallet!

Acknowledgements

Internet Suspend/Resume and OpenISR are trademarks of Carnegie Mellon University. All unidentified trademarks mentioned in this paper are properties of their respective owners.

References

[1] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A. Xen and the Art of Virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (Bolton Landing, NY, USA, 2003).

[2] Caceres, R., Carter, C., Narayanaswami, C., Raghunath, M. Reincarnating PCs with Portable SoulPads. In *Proceedings of Mobisys 2005: the Third International Conference on Mobile Systems, Applications and Services* (Seattle, WA, June 2005).

[3] CAMPBELL, RICHARD. *Managing AFS: The Andrew File System*. Prentice Hall, 1998.

[4] GRINZO, L. Getting Virtual with VMware 2.0. *Linux Magazine* (June 2000).

[5] JOHNSON, D. Ajax: Dawn of a new developer. *Java World* (October 2005).

[6] KOZUCH, M., HELFRICH, C., O'HALLARON, D., SATYANARAYANAN, M. Enterprise Client Management with Internet Suspend/Resume. *Intel Technical Journal 8*, 4 (November 2004).

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

[8] KOZUCH, M., SATYANARAYANAN, M., BRESSOUD, T., HELFRICH, C., SINNAMOHIDEEN, S. Seamless Mobile Computing on Fixed Infrastructure. *IEEE Computer 37*, 7 (July 2004).

[9] LAGAR-CAVILLA, A., TOLIA, N., BALAN, R., DE LARA, E., SATYANARAYANAN, M., O'HALLARON, D. Dimorphic Computing. Tech. Rep. CMU-CS-06-123, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, April 2006.

[10] MORRIS, J. H., SATYANARAYANAN, M., CONNER, M.H., HOWARD, J.H., ROSENTHAL, D.S. AND SMITH, F.D. Andrew: A Distributed Personal Computing Environment. *Communications of the ACM 29*, 3 (March 1986).

[11] NATH, P., KOZUCH, M.A., O'HALLARON, D.R., HARKES, J., SATYANARAYANAN, M., TOLIA, N., TOUPS, M. Design Tradeoffs in Applying Content Addressable Storage to Enterprise-Scale Systems Based on Virtual Machines. In *Proceedings of the USENIX Technical Conference* (Boston, MA, June 2006).

[12] RICHARDSON, T., STAFFORD-FRASER, Q., WOOD, K. R., AND HOPPER, A. Virtual Network Computing. *IEEE Internet Computing 2*, 1 (Jan/Feb 1998).

[13] SATYANARANYANAN, M. KOZUCH, M.A., HELFRICH, C.J., O'HALLARON, D. Towards Seamless Mobility on Pervasive Hardware. *Pervasive and Mobile Computing 1*, 2 (2005), 157–189.

[14] SATYANARAYANAN, M. The Evolution of Coda. *ACM Transactions on Computer Systems 20*, 2 (May 2002).

[15] SCHWANN, P. Lustre: Building a File System for 1,000-node Clusters. In *Proceedings of the 2003 Linux Symposium* (Ottawa, Canada, July 2003).

[16] STEINER, J.G., NEUMAN, C., SCHILLER, J.I. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Conference Proceedings* (Dallas, TX, Winter 1988).

[17] TOLIA, N., ANDERSEN, D.G., SATYANARAYANAN, M. Quantifying Interactive User Experience on Thin Clients. *IEEE Computer 39*, 3 (March 2006).

[18] TOLIA, N., HARKES, J., KOZUCH, M., SATYANARAYANAN, M. Integrating Portable and Distributed Storage. In *Proceedings of the 3rd Usenix Conference on File and Storage Technologies* (San Francisco, CA, March 2004).

[19] VENEZIA, P. Sun whips its Sun Ray thin client into better shape. *InfoWorld* (August 2006).