Pocket ISR: Virtual Machines Anywhere

Benjamin Gilbert, Adam Goode, Mahadev Satyanarayanan
Carnegie Mellon University

March 2010
CMU-CS-10-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was supported by the National Science Foundation (NSF) under grant number CNS-0509004 and by an IBM Open Collaborative Research grant. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, IBM, or Carnegie Mellon University.

Keywords: Internet Suspend/Resume, OpenISR, USB key, Live CD, Linux, Fedora, VirtualBox, virtual machines

Abstract

The Internet Suspend/Resume® system provides access to a user's computing state anytime and anywhere — so long as there's an ISR client handy. We introduce Pocket ISR, a Linux distribution which can be installed on a USB key and booted on any borrowed PC to quickly provide a complete ISR client without modifying the borrowed machine. In order to ensure adequate I/O performance, Pocket ISR noninvasively uses free disk space on the host computer to store parcel data. Pocket ISR is easy to set up, can automatically update itself to the newest version, and allows an ISR user to achieve the promise of ISR: that personalized computing can be done at any Internet-connected computer, anytime and anywhere.

1 Introduction

The Internet Suspend/Resume® system [19] allows users to suspend their computing session at one computer and seamlessly resume it at another. A user stores her computing state — applications, documents, and settings — in a virtual machine called a *parcel*, which is stored on an Internet-connected server. To use the parcel, she directs ISR client software to *check out* the parcel from the server and *resume* it. This causes the suspended virtual machine to be downloaded from the server to the client computer and launched within a virtual machine monitor, allowing the user to perform her everyday computing tasks. The parcel executes on the client, not on the server; thus, ISR does not suffer from the poor interactive responsiveness endemic to thin clients [22]. When the user is ready to *suspend* her parcel, she can direct the ISR client to *check in* the parcel, which uploads it back to the server. When checking out a parcel, ISR only needs to download its suspended memory image; the parcel's virtual disk is divided into chunks which are fetched on demand. When checking in, only modified data is uploaded to the server. Parcel data is compressed to save bandwidth and encrypted for security.

Thus, the Internet Suspend/Resume system, via the OpenISR® software suite, allows a user to obtain access to her computing state from any Internet-connected computer. As a result, she no longer needs to carry a laptop in order to perform day-to-day computing tasks. Any borrowed desktop or laptop computer can run an OpenISR client and resume her parcel. But what if the borrowed computer does not have the client installed? The user may not have the ability or authorization to install it. The client software currently does not support Windows, so installing it on the borrowed computer may be impractical. And even if the client is already installed, the user may not trust it: the borrowed computer may be infected with keylogging or data-stealing malware, making it unsafe for the user to resume her parcel on the system.

One solution is to allow the user to carry the OpenISR client with her on a portable device. Rather than using the computer's pre-installed operating system and OpenISR client, the user could simply boot a so-called "live" Linux system from a portable USB storage device (a "USB key"). In one step, the user would thus gain access to a complete operating environment, the OpenISR client, and a suitable virtual machine monitor. Provided that the computer had access to the Internet, the user would then have everything she needed to resume her parcel. Since the operating system and applications already installed on the computer would be ignored, the user need not fear any malware it may contain; in the absence of malicious hardware such as a hardware keylogger, her data would be safe. Once the user had finished her session and checked in her parcel, she could simply shut down the computer and unplug the USB key. The owner of the borrowed computer would need to trust that the user, and the software installed on her USB key, were not malicious, but need not fear accidental breakage, since the operating system and applications on the computer would remain untouched.

This solution has one flaw: it leaves the OpenISR client no adequate place to store a parcel's data while it is running. As Table 1 shows, running a parcel requires significant amounts of temporary storage space

Component	Approximate Size
Encrypted metadata	5.5 MB
Encrypted and compressed memory image	250 MB
Working copies of metadata	11 MB
Working copy of memory image	512 MB
Disk chunk download cache	2 GB
Disk chunk working cache	2 GB
Upload cache	500 MB
Total	$\sim$ 5.3 GB

Table 1: Typical temporary storage requirements for resuming and checking in a parcel with a 512 MB memory image and an 8 GB virtual disk.

on the client. This volume of data cannot be held in RAM, and storing it on the USB key is infeasible: the poor write performance of consumer USB flash devices would render ISR too slow to be usable. The client has one recourse: borrow unused disk space from the host computer's internal hard drive.

This report presents Pocket ISR, an encapsulation of the OpenISR client into a Linux live image that safely uses the hard disk of the host computer for temporary storage of parcel data. Pocket ISR allows the user to turn any PC into an OpenISR client in as little as 60 seconds, without installing anything on the PC and without leaving any trace afterward. It provides performance comparable to an OpenISR client installed directly on the host system, regardless of the performance of the USB key it booted from. Pocket ISR can easily be installed onto a USB key from either Linux or Windows, without disturbing data already stored on the key. It can even be run from CD with no additional performance overhead after it has booted. And, if installed on a USB key, Pocket ISR can automatically update itself to the latest version with only two clicks. The Pocket ISR tools and compilation are released under version 2 of the GNU General Public License [5], and Pocket ISR is available for download from the ISR website [8].

The remainder of this report is organized as follows. Section 2 describes the construction of the Pocket ISR live image. Section 3 presents the means by which Pocket ISR obtains storage for parcel data. Section 4 explores future extensions to Pocket ISR. Section 5 discusses related work, and Section 6 concludes the report.

2 Pocket ISR Boot Image

The Pocket ISR Linux image is derived from Fedora Linux [17] and uses the same tools used by the Fedora Project to create live versions of Fedora. These tools allow the ISR project to generate and distribute a single ISO9660-format image file which can be burned directly to a CD or written to a USB key with the help of a straightforward conversion tool. Pocket ISR includes many customizations on top of the basic Fedora Live CD infrastructure.

2.1 Fedora Live Infrastructure

Distributing a bootable operating system image that can be run from a USB key is challenging for several reasons. The end user must be able to copy the image to the USB key using whatever tools she has; if she runs Linux, then `dd` at least is available, but Windows comes with no such tool. In addition, the size of her USB key is not known in advance, so the image's partition table will certainly not match her storage device. As a result, if the user wants to use any extra space on the USB key for other data — including ISR parcel data — she must manually adjust the partition table and resize or create a partition. If this is done incautiously, it may be necessary to rerun the boot loader setup tool to account for the partition changes. The result: the user must navigate a long list of instructions in order to set up her USB key. The cost of a mistake is high: she might repartition the wrong disk! This approach may work for highly-motivated users, but it provides too high of a barrier for more casual users of ISR.

The Fedora Live infrastructure takes a different approach. The bootable root filesystem is stored as a regular ext4 filesystem, but in an image file rather than a partition. To save space, that file is stored in a SquashFS [11] compressed read-only filesystem, which itself is stored in an image file. The SquashFS image, the kernel and `initramfs` images, and a boot loader are then placed in an ISO9660 live image which can be burned directly to a CD. If the user wants a CD-based Pocket ISR solution, nothing further is necessary: she can use standard CD-burning software to record the image to a CD which can boot any x86-based PC.

However, a CD is less portable than a USB key and booting from CD is slower than booting from USB, so most users will want to install Pocket ISR on a USB key. To do this, a user simply gives the ISO image file to a tool which unpacks the kernel and filesystem image onto her existing USB key and configures the boot loader (Figure 1). Because the boot loader supports the FAT filesystem, the USB key can remain in this

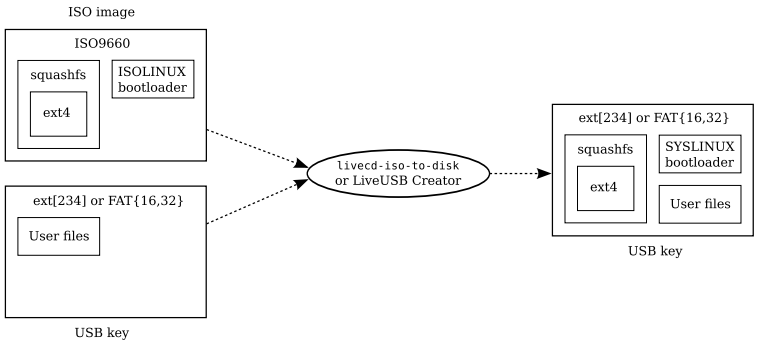


Figure 1: Installing Pocket ISR on a USB key.

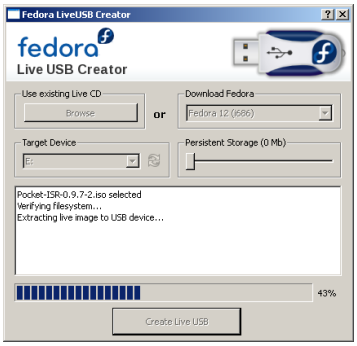


Figure 2: LiveUSB Creator.

standard format, allowing the user to store other data in the unused space. The USB key itself does not need to be repartitioned or reformatted, so existing data on the USB key is preserved.

For a user, the process of creating a Pocket ISR USB key is therefore reduced to two steps: downloading an ISO image file and running a program to set up the USB device. On Linux, she can run the `livecd-iso-to-disk` command-line tool, which is included in the ISO image in case she does not already have a copy. On Windows, she can download the LiveUSB Creator tool, which provides a simple GUI to perform the same task (Figure 2).

2.2 Customizations

Pocket ISR is implemented as a custom live image built using the Fedora `livecd-creator` tool and derived from the official LXDE spin of Fedora Linux. The ISO image is generated by an automated tool driven by a Fedora "kickstart" file, which lists the software packages that should be installed in the live image and the customizations that should be applied. Naturally, these customizations include installation of an OpenISR client and the Pocket ISR tools. In addition, several other modifications are applied:

Download size optimizations. In order to reduce the download size of the live image, Pocket ISR incorporates several modifications to the Fedora Live CD build. Pocket ISR ships the Lightweight X11 Desktop Environment [13] rather than the larger GNOME desktop. It includes only those desktop applications which are necessary in order to run Pocket ISR, which in practice is little more than a terminal program and a virtual machine monitor. (To allow Pocket ISR to be used on networks with web-based captive portals, it also includes the minimal Midori web browser [4].) Unnecessary configuration tools (e.g., for configuring printing or enabling a firewall) have been removed.

For additional space savings, several other optimizations are performed. Non-English fonts and locale data are removed, as well as documentation from packages not specific to Pocket ISR. In addition, development packages such as GCC and Make are removed. These are required early in the image construction process to compile kernel modules for VirtualBox and for the OpenISR client, but are no longer necessary afterward.

Addition of VirtualBox. Pocket ISR includes the VirtualBox virtual machine monitor [14]. VirtualBox is available in two editions: the closed-source version packaged by Oracle and an open-source version, known as VirtualBox OSE, which is released under the GNU General Public License [5]. Oracle does not permit redistribution of the closed-source version, and the open-source version is not packaged in Fedora due to its dependence on third-party kernel modules. Pocket ISR therefore includes a custom build of VirtualBox OSE packaged specifically for the project. To save space, this build does not include the VirtualBox Guest Additions, which are normally only required when first configuring a

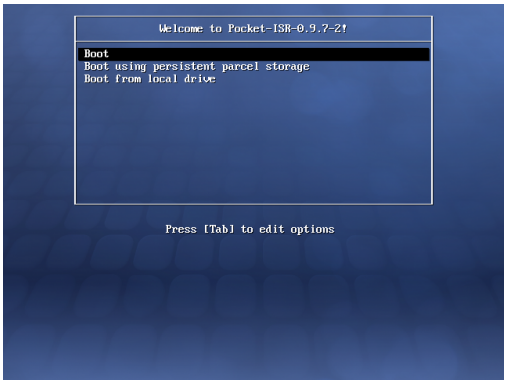


Figure 3: Pocket ISR boot menu.

new parcel. Users setting up a newly-created parcel using Pocket ISR must therefore obtain the Guest Additions from another source.

Every major release of VirtualBox includes incompatible changes to the XPCOM API upon which the OpenISR client depends. As a result, the latest release of the OpenISR client often does not support the most recent release of VirtualBox. Pocket ISR includes the most recent version of VirtualBox OSE that is supported by ISR, not the most recent version released by Oracle.

Modification of boot menu. Pocket ISR includes several changes to the boot loader menu displayed at startup (Figure 3). Fedora live images normally do not show the menu at all unless a key is pressed, and will automatically boot the live image after 10 seconds. Pocket ISR, however, always shows the menu and does not have a default: the user must explicitly choose to boot into Pocket ISR. Menu items allowing the user to verify the checksum of the live image or to test system RAM are removed. Most importantly, the Pocket ISR boot menu allows the system to boot in one of two different storage modes: persistent mode, described in Section 3.1, or transient mode, described in Section 3.2.

Special support for OpenISR client configuration file. An ISR user typically stores user-specific settings, such as her preferred ISR server and user ID, in an `.openisrrc` file in her home directory. Because Pocket ISR does not use a consistent home directory from boot to boot, another mechanism must be provided for configuring the OpenISR client. Therefore, during the boot process, Pocket ISR looks for an `.openisrrc` file at the root of the CD or USB key that was used to boot the system. If this file exists, it is copied into the user's home directory. The user can therefore include an ISR configuration file on her CD or USB key, and this configuration will be used on every boot.

Removal of Fedora branding. Because Pocket ISR includes software not part of Fedora Linux, the Fedora trademark guidelines [18] require the removal of Fedora logos and other branding from the Pocket ISR image. This is accomplished by removing the RPM packages containing Fedora branding and replacing them with generic versions provided by the Fedora project. Unfortunately, the generic version of the graphical boot progress bar prominently features a cartoon drawing of a hot dog (Figure 4), which was not considered appropriate for Pocket ISR. Therefore, Pocket ISR disables graphical boot in favor of the text-based progress bar shown in Figure 5.

2.3 Updating Pocket ISR

The Fedora live infrastructure, like most live Linux distributions, provides the appearance of read/write access to the root filesystem. This is convenient because it allows the user to install and use software that was not originally included with the live system. Since the root filesystem is actually a compressed, read-only image, it cannot actually be modified; changes are directed to a transient overlay stored in RAM or a

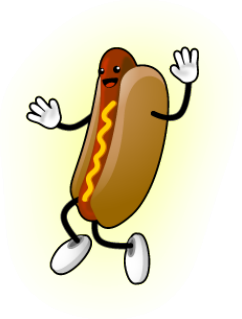


Figure 4: Hot dog with arms, legs, and mustard.

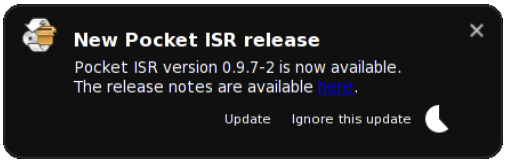
Figure 5: Boot progress bar.

persistent overlay stored on the boot disk. As a result, live versions of desktop Linux distributions generally disable their software update tools, since updates cannot cleanly be merged with the original image. Live distributions can therefore quickly become out-of-date with respect to security updates and new versions of important software.

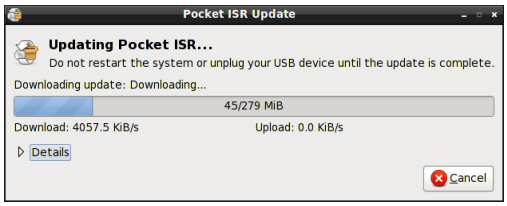
This is a concern for Pocket ISR because it indefinitely extends the period of time over which a particular version of the OpenISR client must be supported. The OpenISR system is still evolving, and the ability to eliminate server support for very old versions of the client remains important. When the OpenISR client is installed on a desktop Linux system, it is automatically kept up-to-date by the Linux distribution's update tool, so the installed base of older OpenISR clients is limited. A user's Pocket ISR installation, however, may be used infrequently — perhaps only when the user is on travel — and is inherently more difficult to update.

Pocket ISR therefore includes an automatic update tool similar to those shipped by desktop Linux distributions. Unlike those tools, which update individual software packages, Pocket ISR Update downloads and installs a completely new version of the live image. This approach is made possible by the unusual properties of Pocket ISR: it is typically booted from a writable disk, but has no dependencies on that disk once it has booted (see Section 3.2.3), and large quantities of scratch space are available for temporary storage of the downloaded image.

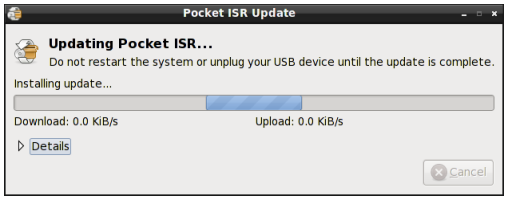
The user interface of Pocket ISR Update is shown in Figure 6. When Pocket ISR boots and obtains a network connection, Pocket ISR Update checks the Pocket ISR website to determine whether a new version



(a) Update notification popup.



(b) Downloading an update.



(c) Installing the update.

(d) Update complete.

Figure 6: Updating Pocket ISR.

is available. If so, it displays a popup (a) and a notification icon which prompt the user to update. If the user agrees, Pocket ISR Update downloads the update (b), writes it to the USB key (c), and prompts the user to restart in order to use the updated version (d).

The implementation of Pocket ISR Update is straightforward: the entire ISO image of a new release is downloaded into temporary storage, verified against its GnuPG [6] signature, and written to the USB key with `livecd-iso-to-disk`. Because Pocket ISR in transient mode runs entirely from RAM, this operation does not interfere with the running system.

A typical Pocket ISR image is 280 MB. In order to reduce the load on the project's web server, Pocket ISR Update uses the BitTorrent protocol [1] via the Rasterbar BitTorrent library [16] to download the ISO image. The torrent uses web seeding [7] to ensure that download is still possible if the client is behind a restrictive firewall or there are not enough peers.

Pocket ISR Update performs several safety checks before consenting to update the system. If Pocket ISR was booted in persistent mode, the updater will refuse to perform the update, since the OS is still depending on the USB key for its root filesystem. If the USB key has been unplugged, update is likewise impossible. Finally, if Pocket ISR was booted from a CD, the updater will still notify the user of an available update but will not offer to update the system, since this is clearly impossible.

3 Parcel Storage

The OpenISR client requires a significant amount of temporary storage space in order to run (see Table 1). Before a parcel can be resumed, its compressed and encrypted memory image must be downloaded and stored to disk, decoded and decompressed, and then stored to disk again. While the parcel is running, disk chunks that are demand-fetched from the server must be stored twice: once into a content-addressable cache to minimize network accesses, and once into a chunk store where it can be modified by subsequent write operations. During checkin, the new memory image must be written out to disk, compressed, encoded, and written out again, and modified disk chunks must be copied into a separate upload cache before being transferred to the server. If the underlying disk does not provide good I/O performance for all of these operations, the user experience afforded by the OpenISR client may be unacceptably poor.

However, most of the data written to disk during an ISR session does not need to persist indefinitely. All of the modified data generated by a running parcel is returned to the server once the parcel is checked in. The only data remaining on the client consists of caches of data also stored on the server; retaining it will improve performance on the next resume but is not otherwise necessary.

Pocket ISR offers the user a choice of two mechanisms for storing parcel data: *transient* and *persistent* parcel storage. When Pocket ISR first starts up, it asks the user which of these mechanisms she wishes to use for the current session (Figure 3). Persistent mode is discussed in Section 3.1, while transient mode is discussed in Section 3.2.

3.1 Persistent Parcel Storage

Persistent storage is the simpler of the two models. It takes advantage of the Fedora live system's "persistent home directory" feature: while initially configuring the USB key, `livecd-iso-to-disk` can create a `home.img` file on the device. This image file contains an ext4 filesystem which is automatically mounted at `/home` when Pocket ISR boots. This approach provides one obvious advantage: because parcel storage is persistent, the user need not check in her parcel before shutting down Pocket ISR, and previously-downloaded chunks are still cached when the system starts up again. However, persistent mode imposes several restrictions which make it impractical for most users:

- Because parcel data is stored directly on the USB boot device, this device must provide sufficient I/O performance for a good interactive experience. In practice, the random write performance of most

USB flash devices is too poor for real ISR usage. USB-connected hard disks and solid-state disks provide sufficient performance, but in a form factor too large for most users to willingly carry.

- As Table 1 shows, resuming a moderately-sized parcel may require gigabytes of free space. However, if the USB key is formatted with the FAT filesystem, the `home.img` file (and thus persistent storage as a whole) is limited to 4 GB. Therefore, for persistent storage to be feasible, the USB device must be formatted with an ext2, ext3, or ext4 filesystem. This requirement limits the user's ability to store non-ISR files on her USB key, since the key can only be read by Linux.

- Because the `home.img` file needs to contain an ext4 filesystem, the USB key can no longer be created on a Windows system.

- This approach requires the boot medium to be writable, so it cannot be used when booting Pocket ISR from CD.

As a result of these restrictions, persistent mode is expected to be useful to very few users and is not the default.

3.2 Transient Parcel Storage

Transient mode, the default Pocket ISR mode, stores parcel data in free disk space on the host computer. Since the computer's primary purpose is almost certainly not to run Pocket ISR, it is important that its stored data not be altered. Therefore, Pocket ISR cannot simply mount the host's filesystems and store files in them: that approach would expose the host's data directly to the Pocket ISR user, increasing the risk of accidental modifications or deletions. It also risks damaging the filesystems due to buggy filesystem drivers, causing further damage to inconsistent or damaged filesystems, or inadvertently enabling backward-incompatible filesystem features. Most importantly, this approach cannot be used if the host's normal operating system is hibernated, since the hibernated image contains cached filesystem state which would no longer be in sync with the actual state of the host's disk.

Pocket ISR therefore takes a lower-level approach. The metadata of each recognized filesystem is scanned to locate unallocated filesystem blocks, which are then combined into a virtual disk. This approach allows Pocket ISR to aggregate free space from multiple host filesystems and safely use it without modifying any host data or metadata.

3.2.1 Collecting Free Disk Space

The task of collecting available disk space into a scratch volume is performed by the `gather_free_space` tool, which is implemented in about 550 lines of C. `gather_free_space` performs the following steps:

1. The `blkid` library from e2fsprogs [24] is used to gather a list of available disk volumes and the types of the filesystems they hold. Because `gather_free_space` is run after the Fedora `initramfs` scripts, Linux software RAID and Logical Volume Manager (LVM) [12] volumes have already been assembled, so the list returned by `blkid` includes volumes managed by these subsystems. The list is then filtered to remove volumes stored on the Pocket ISR USB key itself.

2. Each volume is examined via filesystem-specific libraries. If the volume contains an ext2, ext3, or ext4 filesystem, it is examined using the `ext2fs` library from e2fsprogs. If it contains an NTFS filesystem, it is examined using `libntfs` from ntfsprogs [10]. If it contains a Linux swap partition, it is examined directly by `gather_free_space`. Volumes containing other filesystem types are skipped.

   For each recognized filesystem, `gather_free_space` first checks whether the filesystem appears to be in a consistent state. If the filesystem's metadata indicates that it was not closed cleanly or is

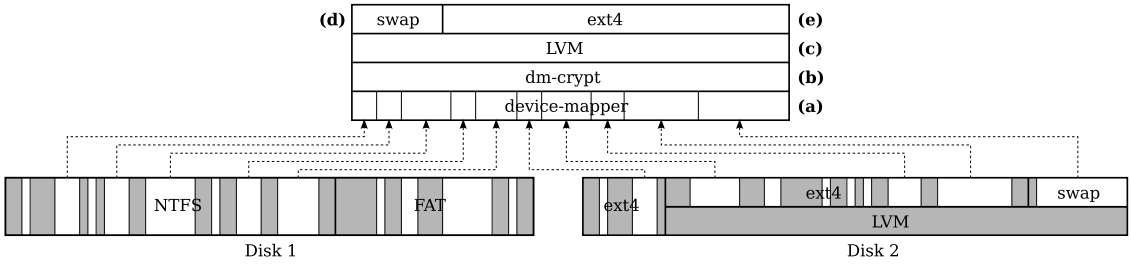


Figure 7: Assembling free disk space.

known to have errors, or if an NTFS or Linux swap partition contains hibernated memory state, the entire volume is skipped. In addition, if any errors are encountered while reading the volume, it is skipped. These tests attempt to ensure that `gather_free_space` will not cause further damage to a host filesystem that it cannot properly handle.

For each volume that passes these tests, `gather_free_space` obtains a bitmap of allocated filesystem blocks. The bitmap is scanned to generate a stream of *extents*, or runs of unallocated blocks. The lengths and locations of the largest 100,000 extents yet encountered are recorded in a binary heap. To improve performance, extents smaller than 4 MB are not recorded; such extents contribute only a small amount of storage, but incur access latencies similar to those of larger extents. If these small extents were included in the scratch volume, ordinary I/O operations could require large numbers of disk seeks, dramatically reducing overall throughput.

3. The Linux device-mapper infrastructure is directed, via the `libdevmapper` library, to create a block device (the *scratch volume*) from the extents recorded in the binary heap. Device-mapper [9] provides a mechanism for creating virtual disks that redirect accesses to other disks in a configurable, table-driven fashion; this is the mechanism that underlies the Logical Volume Manager. Before the list of extents is presented to `libdevmapper`, it is sorted by originating volume and sector offset within that volume. This reduces seeks during streaming accesses to the scratch volume, since its sectors are more likely to be ordered with respect to sectors on the physical disks.

   The device-mapper implements a logical volume as a table of extents. If the host's filesystems were highly fragmented and `gather_free_space` were to include every free extent, the table could grow quite large. The Linux kernel places the table in the `vmalloc` allocation area, which is limited to 128 MB on x86 systems and cannot be swapped to disk. Limiting the table to the largest 100,000 extents ensures that `gather_free_space` creates the largest scratch volume possible without consuming more than a few megabytes of kernel memory.

3.2.2 Using Free Disk Space

`gather_free_space` can create a virtual disk, but cannot make it usable by Pocket ISR. This task is performed by the `early-scratch-setup` initialization script, which runs during boot just before the `livesys` initscript. As shown in Figure 7, `early-scratch-setup` performs the following actions:

1. `gather_free_space` is run to create a scratch volume (a). By default, `early-scratch-setup` imposes a minimum volume size of 2176 MB (2 GB + 128 MB); if `gather_free_space` is unable to create a scratch volume of at least this size, `early-scratch-setup` produces an error and terminates.

2. A `dm-crypt` volume (b) is created which is backed by the scratch volume and encrypted with AES, using the XTS cipher mode and a randomly-generated 256-bit key. This step ensures that sensitive data stored by the OpenISR client, such as parcel decryption keys, cannot be recovered from the disk once Pocket ISR is shut down.

3. A Logical Volume Manager volume group (c) is created which is backed by the dm-crypt volume.

4. A swap partition (d) is created within the volume group. By default, this swap partition is 2 GB in size. The swap partition is formatted and enabled for use.

5. A storage volume (e) is created from the remainder of the space in the volume group. The volume is formatted with an ext4 filesystem and mounted on `/home`. To improve performance, `mkfs.ext4` is run with the `lazy_itable_init` option, causing `mkfs.ext4` to defer initialization of filesystem block groups to a background task which runs when the volume is first mounted.

This entire process takes less than 15 seconds on a 500 GB disk.

`early-scratch-setup` only runs if the `scratch_volume` parameter is specified on the kernel command line, allowing the boot loader to indicate whether the user selected transient or persistent mode at boot time. If this option is specified, `early-scratch-setup` requires that two other kernel command-line options are also provided; these are necessary to disable boot-time mechanisms that interfere with `early-scratch-setup`. The `nopersistenthome` option tells the `livesys` initscript not to mount any existing `home.img` file onto `/home`; this is necessary to allow transient mode to work with USB keys that have been configured to support persistent mode. The `noswap` option is necessary to prevent `livesys` from enabling any swap partitions it might find on the host's disks; since these partitions have already been included in the scratch volume, they cannot also be used as swap space.

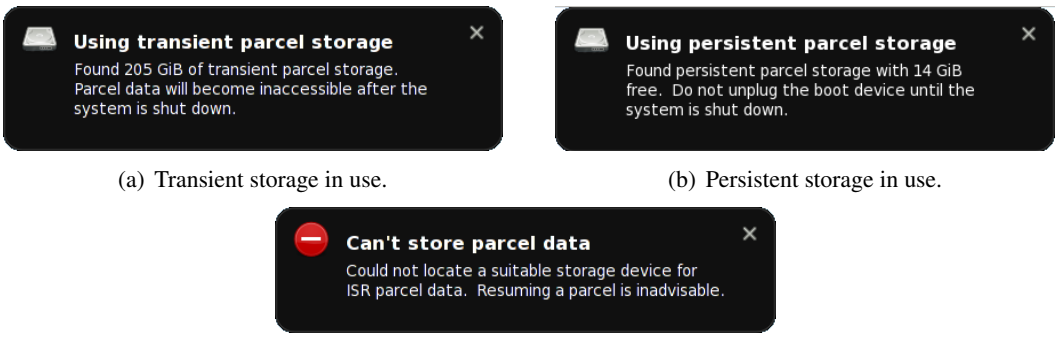
3.2.3 Early Unplug

Transient mode makes it possible to avoid storing parcel data to the USB key, but so far we have assumed that Pocket ISR depends on the USB key for operating system files. However, this dependence should not be necessary: Pocket ISR could copy the entire operating system from the USB key into temporary storage, allowing the user to unplug the USB key immediately after boot is complete. However, doing so poses a problem. By the time the OS initialization scripts are run, the system has already committed to using the root filesystem stored on the USB device, a decision which is very difficult to reverse. The only practical way to select a different root filesystem is to modify the `initramfs`, a miniature filesystem loaded into memory by the boot loader and tasked with obtaining access to a suitable root filesystem. The `initramfs` on Fedora systems is created by the modular Dracut tool. In principle, a Dracut plugin could be constructed which ran `early-scratch-setup`, copied the root filesystem into the resulting volume, and booted from it. However, the `initramfs` is an unforgiving development and test environment, increasing the implementation cost of such an approach.

Fortunately, there is a cleaner alternative. The Dracut module responsible for booting live systems already supports a `live_ram` kernel command-line parameter, which causes the entire SquashFS image to be copied from the boot medium into RAM before the system is booted. (The option was originally provided to improve performance when booting from CD.) Pocket ISR enables this option when booting in transient mode; once boot is complete, the system no longer depends on the USB key, which can be unplugged. Even though the entire operating system is loaded into RAM, memory pressure in the system is not greatly increased: under low-memory conditions, the kernel can migrate unused parts of the operating system image to the swap partition created by `early-scratch-setup`.

3.3 Notifying the User

The process of loading the operating system into RAM and building the transient storage volume, if applicable, typically takes less than 30 seconds, and the entire boot process less than 60. As a result, Pocket ISR does not need to display its own progress indicator while these tasks are running; they are all hidden under the system's boot progress bar (Figure 5). However, the user would be aided by feedback indicating whether sufficient parcel storage was found, and thus whether it is safe to resume her parcel. Therefore, once the



(a) Transient storage in use.

(b) Persistent storage in use.

(c) No parcel storage available.

Figure 8: Parcel storage notifications.

boot process completes and the desktop environment is running, the user is shown a popup message like those in Figures 8(a) and (b). The message indicates what type of parcel storage is in use, gives the amount of disk space available, and reminds the user of the consequences of her chosen storage mode. If transient mode failed to locate sufficient storage, or if persistent mode failed to locate the `home.img` file, a warning (c) is shown instead.

4 Future Work and Extensions

This section discusses several potential improvements to the Pocket ISR system.

4.1 Reducing Pocket ISR Download Size

The current version of the `livecd-creator` tool creates the SquashFS image using gzip compression. Development is ongoing on a version of SquashFS which supports the LZMA compression algorithm, which typically provides significantly better compression ratios. Once LZMA support is integrated into the SquashFS tools and supported by `livecd-creator`, it will be used in Pocket ISR to reduce the size of the Pocket ISR ISO image. Tests with the Pocket ISR 0.9.7-2 release show that switching to LZMA compression could reduce the ISO image size by more than 40 MB.

4.2 Supporting Additional Virtual Machine Monitors

The current version of Pocket ISR supports only the VirtualBox virtual machine monitor. To improve compatibility with existing parcels, Pocket ISR should strive to include all of the VMMs that are supported by the OpenISR client and can legally be redistributed. At present, the only other VMM that meets these criteria is KVM. As of Fedora 12, VirtualBox cannot run while the KVM kernel modules are loaded due to conflicts over the virtualization hardware, preventing Pocket ISR from including both VMMs. This problem is resolved starting with Linux kernel 2.6.33, which will be included in Fedora 13. The first version of Pocket ISR to be derived from Fedora 13 will therefore include support for KVM.

4.3 Supporting Additional Host Filesystems

`gather_free_space` supports collecting free extents from all commonly-used filesystems except FAT and HFS+. Unlike with the ext2 family and NTFS, no widely-used libraries exist for reading these filesystem types. Several relatively-obscure libraries exist for reading the FAT filesystem, and one of these might be used; however, the internal hard disks of modern computers are not commonly formatted with FAT. The absence of HFS+ support, however, prevents transient mode from functioning on the vast majority of x86-based Macintosh systems. The HFS+ format is documented by Apple and supported by the Linux kernel, but no user-space libraries support it. Proper support of HFS+ in Pocket ISR would therefore likely require `gather_free_space` to directly parse HFS+ data structures.

4.4 Improving User Feedback

Once Pocket ISR has finished booting in transient mode, it shows the user a notification giving the total amount of free disk space that was located (Figure 8(a)). This display is sufficient for day-to-day use of Pocket ISR, but is not sufficient to help the user debug problems. More detailed information is printed by `gather_free_space` as it runs, but this output is normally hidden under the boot progress bar. The user can press the Escape key during boot to see this information, but this is no longer possible once Pocket ISR has finished booting. In addition, debugging information explaining why certain volumes were skipped entirely (for example, they are damaged or contain a Windows hibernate file) is not printed by default.

It may be useful to log the output of `gather_free_space` to a location which could later be examined by the user to diagnose problems. A graphical tool could read the logging output and visually display the filesystems on the host, the quantity of free space obtained from each, and any errors that were encountered while reading the filesystems.

4.5 Caching Parcel Data on the USB Key

An installation of Pocket ISR onto a portable USB storage device consumes less than 300 MB, but today's inexpensive USB keys are significantly larger. Pocket ISR should allow the user to use the remaining space as a lookaside cache [23] for parcel data. This is already supported when Pocket ISR is booted in persistent mode, but as explained in Section 3.1, persistent mode is not the normal usage model. Supporting lookaside caching in transient mode would require several changes to Pocket ISR and the OpenISR client:

- Lookaside caching in transient mode would require the OpenISR client to support more than one cache: one on the USB key and one in transient storage. The client would need to be extended to support multiple caches and to automatically detect and use any cache present on the USB key.
- Any on-key cache supported in transient mode would need to be stored directly in the filesystem of the USB key. Since the USB key is very likely formatted with the FAT filesystem, individual files in the lookaside cache cannot be larger than 4 GB. Meeting this requirement would require changes to the client's existing lookaside cache, which stores all disk chunks in a single large file.
- Once the user has created a lookaside cache on her USB key, the same cache should be usable under other circumstances. For example, if the key is booted in persistent mode, ISR should use the same cache as in transient mode. In fact, any OpenISR client, including clients not running within Pocket ISR, should be able to locate and use lookaside caches on portable USB devices. Therefore, the lookaside cache format and implementation should not be specific to Pocket ISR, but should be a generic mechanism supported by all OpenISR clients.
- Currently, once Pocket ISR has finished booting in transient mode, the user may unplug her USB key at any time. Preserving this feature would require the OpenISR client to gracefully handle the disappearance of a lookaside cache while it is in use; in this case, the client should simply continue to the next available cache, or to the server, to obtain the necessary parcel data.

4.6 Integrating Features from Horatio

Pocket ISR could serve as a starting point for the integration of concepts from Horatio [20] into the OpenISR system. Some parts of the Horatio design are already in place: Pocket ISR can be booted from a smart phone that presents itself as a USB storage device, and checked-out parcels can already be stored on the boot device if Pocket ISR is booted in persistent mode. However, the flash storage in current smart phones is unlikely to provide sufficient performance for persistent mode. Opportunities for further integration of Horatio concepts include:

- When Pocket ISR is booted from a mobile phone, it could use the mobile phone's Internet access to communicate with the OpenISR server.
- The OpenISR system could be extended to support copying of existing checkouts between devices, such as between a desktop OpenISR client and Pocket ISR, or between Pocket ISR's transient storage and the USB key. This would allow the user to store modified parcel state back to Pocket ISR when insufficient bandwidth is available to upload it directly to the server. As with the lookaside cache changes proposed in Section 4.5, this change would require modifications to the client's parcel storage format to work around the 4 GB file size limit imposed by the FAT filesystem.
- Once Pocket ISR has been shut down, an application running on the smart phone could use the phone's Internet access to upload parcel state back to the OpenISR server. Likewise, the application could refresh the phone's lookaside cache in the background, ensuring that the complete contents of the parcel are cached.
- Pocket ISR could continue to upload parcel data even after the data has been copied to the phone and the phone's USB connection has been unplugged. This would allow the user to depart while Pocket ISR finishes checking in the parcel (and presumably shuts down afterward), since if Pocket ISR is unable to complete the upload, the user would retain the copy of their parcel state on their mobile phone. Additionally, Pocket ISR and the phone might cooperate to upload parcel state over their respective Internet connections.

4.7 Improving Extent Selection

The extent-selection algorithm could be further improved to account for the user's disk space requirements. On disks with plenty of free space available, the seek overhead imposed by including smaller extents in the device-mapper table might not be worth the small amount of additional free space gained, even though the resulting table size would be sufficiently small. Furthermore, it may be preferable for `gather_free_space` to collect extents only until the resulting volume reaches a certain size, in order to avoid the overhead of scanning and creating filesystem metadata for storage the user will never actually need. On disks that are nearly full, `gather_free_space` may need to include every available extent, even though the resulting seek overhead might be extreme.

4.8 Incrementally Adding Free Disk Space

Currently, Pocket ISR discovers all available free space on the host system in a single operation. Under some circumstances, the user might want to add an additional pool of free space after boot. For example, if the user needs more disk space than expected, or if the host system does not have enough available free space, she might plug in an external disk. The additional free space could be added to transient storage by creating an additional scratch volume and associated `dm-crypt` volume, adding the latter to the LVM volume group, resizing the LVM storage volume, and performing an online resize of the ext4 filesystem it contains.

4.9 Improving Extent Ordering in the Scratch Volume

Currently, extents in the device-mapper table are sorted by origin volume and offset within that volume. However, `gather_free_space` does not have enough knowledge of the disk's partition table to ensure that volumes are listed in the order they occur on disk. As a result, large seeks might be required when streaming data across extents spanning a volume boundary. In addition, extents originating from LVM or RAID volumes might have a complex relationship to their underlying disk partitions. If `gather_free_space` were able to parse the partition tables, device-mapper tables, and RAID metadata underlying the volumes it scanned, it could more effectively sort extents to reduce unnecessary seeks.

4.10 Reducing the Bandwidth Requirements of Pocket ISR Update

At present, Pocket ISR Update downloads the entire ISO image of a new release before updating the USB key. However, the difference between the current and new releases may be quite small. It should be possible to download only the delta between the Pocket ISR image already stored on the USB key and the new version, dramatically reducing download size.

Doing so is not straightforward, however, due to the process used to create a Pocket ISR image. As explained in Section 2.1, the distributed ISO image is not itself written to the USB key. The majority of the distributed data is contained in a SquashFS image file; since it is compressed, this file changes significantly as a result of minor changes in the underlying filesystem data. For SquashFS filesystems containing many individual files, this problem is not severe: SquashFS resets the compression stream when storing each new file, so files unmodified between two images reduce to the same compressed data and need not be included in a delta. However, the SquashFS image in Pocket ISR stores exactly one file: an ext4 filesystem image. As a result, differencing file-transfer tools like rsync [3] and zsync [15] find only small amounts of similarity, even between two builds of the same version of Pocket ISR.

There are several options for solving this problem:

- Disable generation of the SquashFS layer when the ISO image is built, resulting in a larger image amenable to delta transmission. The transmission savings of the differencing transport might outweigh the loss of file compression, particularly if transport compression was used. Two versions of the ISO image could be offered for download, one compressed for new users (who would need to download the whole image file) and one uncompressed for delta transmission. A client starting with the compressed version of the image could simply extract the ext4 image from the SquashFS and use that as the basis for the differencing transfer. The larger ISO image could then be given to `livecd-iso-to-disk`.

  Unfortunately, file transfer size is not the only consideration. When booting an uncompressed image in transient mode, Pocket ISR would have to load the entire uncompressed image into RAM. The result would be increased Pocket ISR boot times and greater memory pressure on the booted system.

- Perform a differencing transfer of just the embedded ext4 image, but construct the SquashFS and surrounding ISO image on the client. This approach trades CPU time for network bandwidth, but would prevent Pocket ISR from having to load a larger filesystem image into RAM at boot. However, it would also require interacting with `livecd-creator` at a lower level, since the command-line utility does not allow the image build process to be altered in this way.

- Disable generation of the ext4 image and place operating system files directly in the SquashFS. This approach would be ideal for Pocket ISR, since it compresses the data without precluding differencing transfer. Further, the practice of embedding an ext4 filesystem inside a SquashFS filesystem exists to support Linux installers running from the live image, a case unimportant to Pocket ISR. However, this change could require extensive modifications to `livecd-tools` and to the Pocket ISR boot process, including changes to the `initramfs`.

5 Related Work

TransPart [21] is a mechanism for borrowing free space from a host filesystem, for temporary use by a virtual machine running from a USB-booted OS. It is similar in spirit to the `gather_free_space` component of Pocket ISR, though Pocket ISR and `gather_free_space` were developed independently of TransPart. TransPart is a prototype, rather than a production-ready system, and only supports collecting free space from ext2 and ext3 filesystems. TransPart tracks free extents via a database in persistent storage, leading to slower discovery of free extents; for example, TransPart takes 15 times as long as `gather_free_space` to collect 200 GB of free space. This overhead forces TransPart to allocate only the minimum amount of space

required by the virtual machine; in contrast, Pocket ISR always allocates the maximum possible amount of disk space.

The Transparent File System [2] is a modified ext2 filesystem that allows opportunistic applications such as SETI@home to borrow unused disk space without interfering with the primary uses of the disk. Unlike Pocket ISR, TFS is intended to allow such applications to run concurrently with normal use of the computer and within the same instance of the operating system. Because this usage model is more challenging than the one supported by Pocket ISR, the design of TFS is correspondingly more complex. In addition to the usual filesystem objects, TFS supports a new type of file, the "transparent file," for which it provides a weaker persistence guarantee: transparent files may disappear at any time if overwritten by regular data, but otherwise have roughly the semantics of a regular file. This guarantee is both too strong and too weak to be useful for Pocket ISR's transient mode. The OpenISR client cannot tolerate the spontaneous disappearance of stored parcel data; conversely, if Pocket ISR stored parcel data in TFS and the computer were abruptly powered off, the data would remain accessible in the host's filesystem after boot. Since TFS does not provide encryption, sensitive data would thus be exposed to the owner of the computer. Using TFS with Pocket ISR would require mounting the host's filesystem, exposing the host's files within the live environment. Finally, unlike `gather_free_space`, TFS is tied to a specific on-disk format which is incompatible with existing production filesystems (including ext2), making it unsuitable for borrowing free space from an arbitrarily-configured computer.

6 Conclusion

We presented Pocket ISR, a live Linux distribution that allows an ISR user to convert a borrowed PC into an ISR client in less than a minute. Pocket ISR uses free disk space on the host computer to securely store temporary parcel data, providing good performance without modifying any of the host's data and without interfering with a hibernated host. Alternatively, if the user has suitable hardware, Pocket ISR can store parcel data directly on a USB storage device and ignore the host's disks entirely. To ease upgrades, Pocket ISR can automatically update itself to the newest release.

Pocket ISR has the potential to change the way people use the OpenISR system. At present, the absence of a large installed base of OpenISR clients forces an ISR user to resume her parcels only on computers she uses frequently, carrying her own laptop when these machines are not accessible. This need to carry a dedicated computer negates much of the promise of ISR. Pocket ISR solves this problem by replacing the dedicated laptop with just another key on the user's keyring. With that key, and any borrowed hardware, an ISR user can obtain easy and responsive access to her parcel anytime and anywhere.

Acknowledgments

The value of embedding a bootable ISR client in a portable USB storage device was first pointed out by David O'Hallaron, who also proposed the name "Pocket ISR."

References

[1] BITTORRENT, INC. BitTorrent. `http://www.bittorrent.com/`, 2010.

[2] CIPAR, J., CORNER, M. D., AND BERGER, E. D. Contributing Storage Using the Transparent File System. *ACM Transactions on Storage 3*, 3 (2007).

[3] DAVISON, W. rsync. `http://samba.anu.edu.au/rsync/`, 2009.

[4] DYWAN, C. Midori. `http://www.twotoasts.de/index.php?/pages/midori_summary.html`, 2010.

[5] FREE SOFTWARE FOUNDATION, INC. GNU General Public License, version 2. `http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt`, 1991.

[6] FREE SOFTWARE FOUNDATION, INC. The GNU Privacy Guard. `http://www.gnupg.org/`, 2010.

[7] HEADLIGHT SOFTWARE, INC. HTTP/FTP WebSeeding Method for BitTorrent. http://www.getright.com/seedtorrent.html, 2006.

[8] INTERNET SUSPEND/RESUME. Internet Suspend/Resume. http://isr.cmu.edu/, 2010.

[9] LINUX DEVICE-MAPPER. Device-Mapper Resource Page. http://sources.redhat.com/dm/, 2008.

[10] LINUX-NTFS PROJECT. Linux-NTFS. http://www.linux-ntfs.org/, 2007.

[11] LOUGHER, P. squashfs. http://squashfs.sourceforge.net/, 2008.

[12] LVM2. LVM2 Resource Page. http://sourceware.org/lvm2/, 2008.

[13] LXDE FOUNDATION E.V. LXDE. http://lxde.org/, 2010.

[14] ORACLE CORPORATION. VirtualBox. http://www.virtualbox.org/, 2010.

[15] PHIPPS, C. zsync. http://zsync.moria.org.uk/, 2005.

[16] RASTERBAR SOFTWARE. Rasterbar libtorrent. http://www.rasterbar.com/products/libtorrent/, 2010.

[17] RED HAT, INC. Fedora Project. http://fedoraproject.org/, 2010.

[18] RED HAT, INC. Fedora trademark guidelines. http://fedoraproject.org/wiki/Legal/TrademarkGuidelines, 2010.

[19] 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.

[20] SMALDONE, S., GILBERT, B., BILA, N., IFTODE, L., DE LARA, E., AND SATYANARAYANAN, M. Leveraging smart phones to reduce mobility footprints. In *Mobisys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2009), ACM, pp. 109–122.

[21] SMALDONE, S., HARKES, J., IFTODE, L., AND SATYANARAYANAN, M. Safe Transient Use of Local Storage for VM-based Mobility. Tech. Rep. CMU-CS-10-110, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, March 2010.

[22] TOLIA, N., ANDERSEN, D., AND SATYANARAYANAN, M. Quantifying Interactive Experience on Thin Clients. *IEEE Computer 39*, 3 (Mar. 2006).

[23] TOLIA, N., HARKES, J., KOZUCH, M., AND SATYANARAYANAN, M. Integrating Portable and Distributed Storage. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (San Francisco, CA, March 2004).

[24] TS'O, T. E2fsprogs: Ext2/3/4 Filesystem Utilities. http://e2fsprogs.sourceforge.net/, 2010.