Transient Customization of Mobile Computing Infrastructure

Adam Wolbach, Jan Harkes, Srinivas Chellappa, M. Satyanarayanan

April 2008
CMU-CS-08-117

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Kimberley enables rapid software provisioning of fixed infrastructure for transient use by a mobile device. It uses virtual machine (VM) technology, but avoids the performance challenges of running VMs on resource-poor mobile devices. VM execution only occurs in the infrastructure and never on the mobile device; however, the device may transport and interpret parts of VM state. Kimberley decomposes VM state into a widely-available *base VM* and a much smaller *private VM overlay.* The base is downloaded in advance; the overlay is delivered on demand from the mobile device or under its control from a public web site. We have built a prototype of Kimberley, and our experiments confirm the feasibility of this approach.

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0614679 and CNS-0509004. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or Carnegie Mellon University.

Keywords: virtual machine, VM, VirtualBox, cyber foraging, composable computing, OpenDiamond, Nokia N810 tablet, large displays, 802.11, service discovery, Avahi

1 Introduction

Usability often suffers when a mobile device is optimized for size, weight and energy efficiency. On a hand-held device with a small screen, tiny keyboard and limited compute power, it is a challenge to go beyond a limited repertoire of applications. A possible solution to this problem is to leverage fixed infrastructure to augment the capabilities of a mobile device, using techniques such as dynamically composable computing [11] or cyber foraging [1, 5, 2]. For this approach to work, the infrastructure must be provisioned with exactly the right software needed by the user. This is unlikely to be the case everywhere, especially at global scale.

This paper describes *Kimberley,* a system for rapid software provisioning of fixed infrastructure for transient use by a mobile device. Kimberley uses virtual machine (VM) technology, but avoids the performance challenges of running VMs on resource-poor mobile devices. It is thus fundamentally different from approaches such as the Internet Suspend/Resume® system [6, 8] that rely on VM migration. In Kimberley, VM execution only occurs in the infrastructure and never on the mobile device; however, the device may transport and interpret parts of VM state.

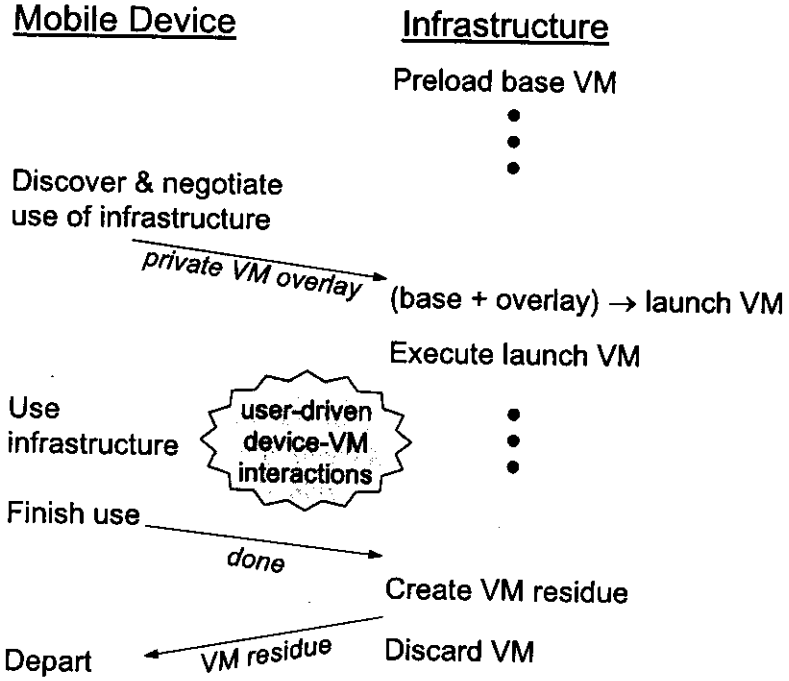


Figure 1: Kimberley Timeline

Kimberley decomposes VM state into a widely-available *base VM* and a much smaller, possibly proprietary, *private VM overlay.* These two components are delivered to the site being provisioned in very different ways. The base VM is downloaded by the infrastructure in advance from a publicly accessible web site. The private VM overlay is delivered to a server in the infrastructure just before use, either directly from the mobile device or under its control from a public web site. In the latter case, encryption-based mechanisms can be used to ensure the integrity and privacy of the private VM overlay. Once obtained, the overlay may be optionally cached for future reuse. The server applies the overlay to the base to create and execute a *launch VM.* When the user departs,

this VM is terminated and its state is discarded. In some cases, a small part of the VM state may be returned to the mobile device; this is referred to as the *VM residue*. Figure 1 shows a typical Kimberley timeline.

We anticipate that a relatively small number of base VMs (perhaps a dozen or so releases of Linux and Windows configurations) will be popular worldwide in mobile computing infrastructure at any given time. Hence, the chances will be high that a mobile device will find a compatible base for its overlays even far from home. The chances of success can be increased by generating multiple overlays, one for each of a number of base VMs. The collection of popular base VMs can be mirrored worldwide, and a subset can be proactively downloaded by each mobile computing infrastructure site.

2 Usage Examples

The two hypothetical examples below illustrate the kinds of usage scenarios we envision for Kimberley.

Scenario 1: *Dr. Jones is at a restaurant with his family. He is contacted during dinner by his senior resident, who is having difficulty interpreting a pathology slide. Although Dr. Jones could download and view a low-resolution version of the pathology slide on his smart phone, it would be a fruitless exercise because of the tiny screen. Fortunately, the restaurant has a large display with an Internet-connected computer near the entrance. It is sometimes used by customers who are waiting for tables; at other times it displays advertising. Using Kimberley, Dr. Jones is able to temporarily install a whole-slide image viewer, download the 100MB pathology slide from a secure web site, and view the slide at full resolution on the large display. He immediately sees the source of the resident's difficulty, helps him resolve the issue over the phone, and then returns to dinner with his family.*

Scenario 2: *While Professor Smith is waiting to board her flight, she receives email asking her to review some budget changes for her proposal. The attached spreadsheet shows a bottom line that is too high. After a few frustrating minutes of trying to manipulate the complex spreadsheet on her small mobile device, Professor Smith looks around the gate area and finds an unused computer with a large display. She rapidly customizes this machine using Kimberley, and then works on the spreadsheet. She finishes just before the final boarding call, and retrieves the modified spreadsheet on to her mobile device. On board, Professor Smith barely has time to compose a reply, attach the modified spreadsheet, and send the message before the aircraft door is closed.*

3 Prototype Implementation

The Kimberley prototype uses a Nokia N810 Internet tablet with a 400MHz TI OMAP processor, 128 MB of DDR RAM and 256 MB flash memory, 2 GB flash internal storage, an attached 8 GB microSD card, and a 4-inch touch-sensitive color display. It supports 802.11b/g and Bluetooth networking, and is equipped with GPS and ambient light sensors. Its software is based on the Maemo 4.0 Linux distribution. The infrastructure machine in our prototype is a Dell Precision 380 desktop with a 3.6 GHz Pentium 4 processor, 4 GB RAM, an 80 GB disk, and a 20-inch 1600x1200 LCD monitor. It runs Ubuntu 7.10, which is based on the Linux 2.6.22-14 kernel. This machine has a 100 Mb/s Ethernet connection to the Internet; it also has 802.11b/g wireless network access via a USB-connected network interface. We describe three aspects of our implementation below.

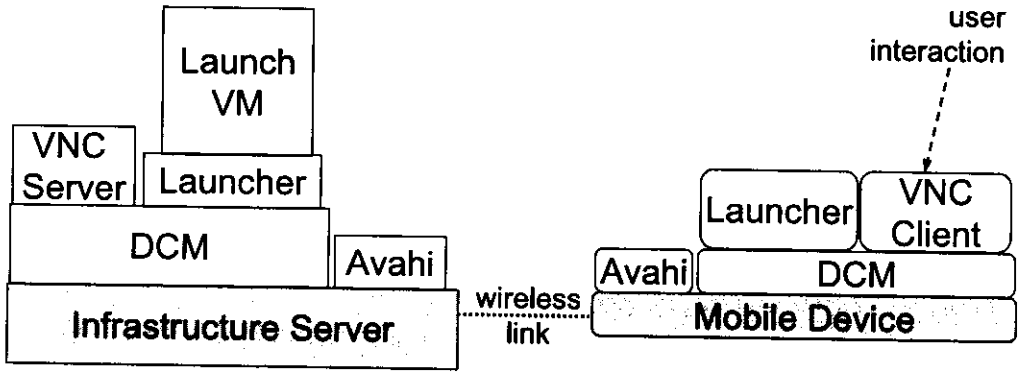


Figure 2: Runtime Binding in Kimberley

3.1 Creating VM overlays

The VMs used in Kimberley are configured using *VirtualBox* [13] on a Linux host. We provide an overlay creation tool called `kimberlize` that is invoked as follows:

```
kimberlize <baseVM> <install-script> <resume-script>
```

`baseVM` is a VM in which a minimally-configured guest OS has been installed. There are no constraints on the choice of guest OS, except that it must be compatible with `install-script` and `resume-script`. The tool first launches `baseVM`, and then executes `install-script` in the guest OS. The result is a VM that has been configured for use by the mobile device. To avoid the run time delays of guest reboot and application launch, `kimberlize` then executes `<resume-script>` in the guest. This yields a VM that can suspended now and then resumed rapidly at run time in a state ready for user interaction. There are no constraints on `install-script` and `resume-script` except for the obvious caveat that they should not halt or crash the guest.

Once the launch VM has been created, `kimberlize` uses the `xdelta` binary differencing tool to obtain the difference between the launch and and post-install memory images. In principle, `xdelta` could also be used to obtain the difference in disk images. However, this is not necessary since VirtualBox already maintains a compact differencing disk. The VM overlay consists of this differencing disk plus the `xdelta`-generated memory difference.

The VM overlay is then compressed using the Lempel-Ziv-Markov algorithm [12], which is optimized for fast decompression at the price of relatively slow compression. This is the appropriate tradeoff for Kimberley because decompression takes place in the critical path of execution at runtime and contributes to user-perceived delay. Further, compression is only done once but decompression occurs each time a mobile device uses infrastructure.

The final step of `kimberlize` is to encrypt the compressed VM overlay using AES-128 private-key encryption. An optional argument to `kimberlize` can be used to specify the private key; otherwise `kimberlize` generates a random key from `/dev/urandom`.

3.2 Binding to infrastructure

Figure 2 shows the key runtime components of Kimberley. The controller of the transient binding between mobile device and infrastructure server is a user-level process called *Device Control Manager (DCM)*. An instance of DCM runs on the device ("device DCM") and on the infrastructure server ("server DCM"). The DCM abstracts service discovery and network management

from the rest of Kimberley. Over time, we envision Kimberley supporting many different types of wireless and wired connections between device and server. While wireless connections are more natural and seamless to the user, there may be situations where the freedom from RF congestion of a dedicated wired link may be preferable for crisp user interaction. The current implementation supports 802.11 (wireless) and USB (wired) connectivity; we expect to add Bluetooth and Ethernet in the future. Note that the mobile device may have a direct connection to the Internet, in addition to the server connection shown in Figure 2.

DCM supports the browsing and publishing of services over the D-Bus interprocess communication mechanism. The use of D-Bus simplifies the implementation of extensibility in service discovery by allowing DCM method calls to be defined in XML files. We currently support the Avahi service discovery mechanism, and plan to support the Bluetooth Service Discovery Protocol (SDP) in the future. It should also be possible to extend Kimberley to use other well-known service discovery mechanisms such as Jini.

The first step in the binding sequence is the establishment of a secure TCP connection using SSL between a device DCM and a server DCM. This secure tunnel is then used by the rest of the binding sequence, which typically involves user authentication and optional billing interaction. Kimberley supports the Simple Authentication and Security Layer (SASL) framework, which provides an extensible interface for integrating diverse authentication mechanisms. After successful authentication, a `dekimberlize` command is executed on the server. This fetches the VM overlay from the mobile device or a Web site, decrypts and decompresses it, and applies the overlay to the base VM. The suspended VM is then launched.

The last step in the binding process is enabling user interaction via VNC [7]. With this setup, all user interactions occur at the mobile device and its screen mirrors a scaled image of the large display on the server. This style of interaction allows the user to remain untethered and to be at some distance from the large server display. However, the ease of interaction is limited by the dimensions of the mobile device. Kimberley also supports interaction through a full-sized keyboard, mouse and other devices attached to the server, as in Scenario 2. This approach may be preferred when the usage context requires more extensive interaction.

3.3 Sharing user data

Some usage contexts require data to be shared between the mobile device and infrastructure server. In Scenario 2, for example, the mobile device could be running Windows CE while the launch VM may run Windows XP. Although versions of the Excel spreadsheet application run in both environments, they are not identical: the mobile version is optimized for interaction in a small device. However, both versions share the same spreadsheet file format. Kimberley needs a mechanism to share such files between device and infrastructure. A file system such as Coda [9] would be an obvious choice. However, this adds to the list of dependencies for making infrastructure compatible with Kimberley, possibly restricting its deployment.

Our solution emulates a large FAT32 floppy disk using a part of the device's flash memory storage. When the mobile device is being used in isolation, this virtual floppy disk is accessible as a mounted file system. When binding to an infrastructure server, Kimberley unmounts this virtual floppy disk, transfers its contents to the server, and then mounts it in the launch VM. When unbinding, Kimberley unmounts the virtual floppy disk, computes the state change since it was mounted, and transmits this difference to the mobile device. The state difference is effectively the

"VM residue" shown in Figure 1. At the device, Kimberley applies the difference and then mounts the virtual floppy disk. The default size of the virtual floppy disk is 20MB, but it can be configured to any size within the storage capacity of the mobile device. Although simple, this solution works well for the usage scenarios that we envision for Kimberley. It is idiot-proof since the virtual floppy disk is never accessible simultaneously on both the mobile device and the VM.

4 Prototype Performance

Our evaluation addresses three questions relative to the use of an infrastructure site that has no cached overlay state:

- How big are typical VM overlays?
- What is the typical startup delay?
- How fast can a user depart?

To answer these questions, we examined six open-source Linux applications. These are listed below, along with their better-known Windows counterparts:

- *AbiWord:* a word processor (Microsoft Word)
- *GIMP:* an image processor (Adobe Photoshop)
- *Gnumeric:* a spreadsheet program (Microsoft Excel)
- *Kpresenter:* a slide tool (Microsoft PowerPoint)
- *PathFind:* a digital pathology tool for whole-slide images based on the OpenDiamond platform [3] and suggestive of Scenario 1.
- *SnapFind:* a tool for examining digital photographs and searching for other similar photographs. It is also based on the OpenDiamond platform.

As a limiting case to explore the intrinsic overhead of Kimberley, we also considered a hypothetical *Null* application by invoking `kimberlize` with empty `install-script` and `resume-script` parameters.

4.1 Overlay Size

Table 1 presents data relevant to the first question. The table shows that the compressed overlay size for these seven applications ranges from 5.9 MB for the Null case to 196.6 MB for PathFind. The uncompressed sizes indicate a typical compression factor of about three. We initially expected overlay creation to be a completely deterministic process, and were therefore surprised to see small variations in overlay sizes for the same application in different experimental runs. This turns out to be an artifact of Virtual Box's differencing disk implementation. Based on more detailed experiments (not described here), we conjecture that VirtualBox asynchronously preallocates real disk space for its virtual disk. The effect is modest, amounting to just a few percent of total overlay size. The last column gives the size of the installation package for an application. This size is loosely correlated with the application's compressed and uncompressed overlay sizes.

Table 2 shows the time taken by `kimberlize` to create overlays. Recall that `kimberlize` sacrifices compression speed for fast decompression at runtime. This accounts for the relatively long times shown in Table 2, ranging from just over a minute for Null to nearly 11 minutes for Gnumeric.

Application	Compressed VM Overlay Size (MB)	Uncompressed VM Overlay Size (MB)	Install Package Size (MB)
AbiWord	119.5 (3.8)	364.2 (19.9)	10.0
GIMP	141.0 (5.2)	404.7 (30.7)	16.0
Gnumeric	165.3 (3.6)	519.8 (24.7)	16.0
Kpresenter	149.4 (7.6)	426.8 (32.2)	9.1
PathFind	196.6 (1.0)	437.0 (2.3)	36.8
SnapFind	63.7 (0.9)	222.0 (3.6)	8.8
Null	5.9 (0.0)	24.8 (0.0)	0.0

Note: Small figures in parentheses are standard deviations

Table 1: VM Overlay and Install Package Sizes

4.2 Startup Delay

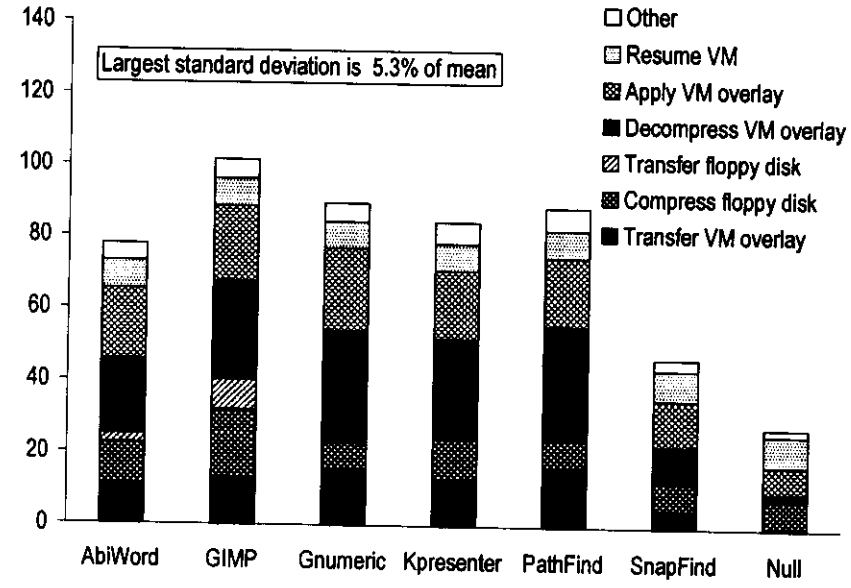
To answer the second question ("What is the typical startup delay?"), we measured the time it takes from a user's initial action to use infrastructure to the point at which he starts interacting with the application. This includes the time taken to transmit the VM overlay and virtual floppy disk, to apply the VM overlay, and to launch the VM.

Application	Elapsed time for `Kimberlize` (seconds)
AbiWord	513 (32.4)
GIMP	527 (39.3)
Gnumeric	652 (36.3)
Kpresenter	548 (45.1)
PathFind	518 (4.3)
SnapFind	277 (3.9)
Null	81 (0.7)

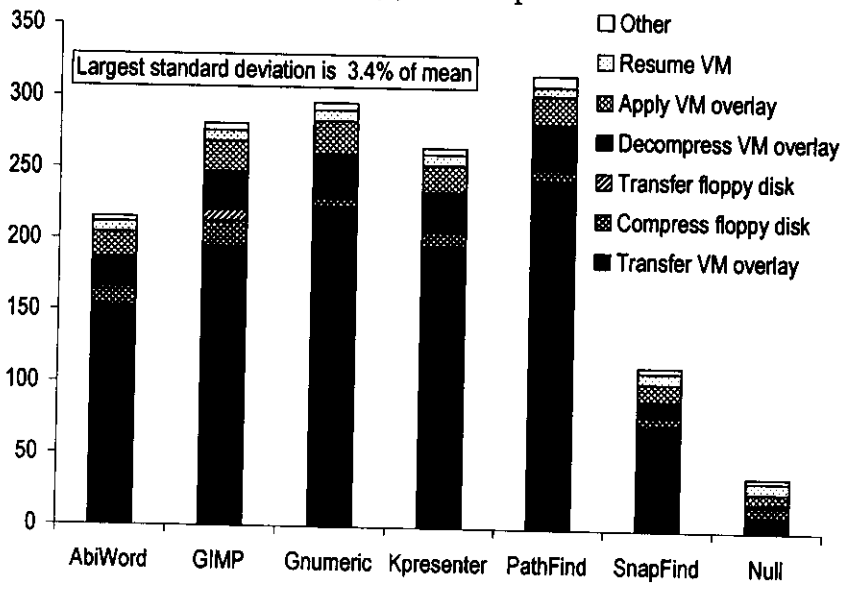
Note: Small figures in parentheses are standard deviations

Table 2: Creation Time for VM Overlays

Our experiments used the same set of seven applications and the VM configuration discussed in Section 4.1. The hardware used in these experiments is the same as that described at the beginning of Section 3. Since network bandwidth is a critical factor that affects overlay transmission time, we explored performance at two different bandwidths. The first bandwidth of 100 Mbps is typical of what one might expect within an enterprise. The second bandwidth of 10 Mbps is suggestive of well-connected public infrastructure. For example, in Scenarios 1 and 2, this might be the bandwidth available from the restaurant or airport gate to the overlay server. Note that bandwidth to the



(a) 100 Mbps



(b) 10 Mbps

Figure 3: Startup Delay in Seconds

overlay server is independent of the wireless connnection between mobile device and infrastructure over which control interactions and transfer of the virtual floppy disk take place.

Figure 3 presents our results for two bandwidths: 100 Mbps and 10 Mbps. Five runs of each experiment were performed. At 100 Mbps, the transfer time for VM overlays only accounts for one-fifth or less of the total startup delay for most applications. Decompressing and applying the VM overlay are the dominant contributors to delay, accounting for roughly half of the total startup delay in most cases. The time to compress the virtual floppy disk is also significant. Improving this quantity is complicated by energy considerations for mobile devices. The total startup delay at 100 Mbps is quite acceptable, ranging from under a minute for SnapFind to just over a minute and a half for GIMP.

At 10 Mbps, VM overlay transfer time dominates. It accounts for roughly three-fourths of the total startup delay for most applications in Figure 3(b). Decompressing the VM overlay and applying it account for roughly one-fourth of the total time in those cases. The total startup delay ranges from roughly 3 to 5 minutes in most cases, which is at the outer limit of acceptability for transient use of infrastructure. This can be improved by using techniques such as data staging [4] to proactively transfer an overlay close to an anticipated usage site. In that case, the startup delay experienced would be closer to the 100 Mbps case. A cached overlay would, of course, also greatly improve startup delay.

4.3 Teardown Delay

Before a user can depart from a site, three steps must occur: the VM must be powered down; the server must generate the VM residue; and the residue must be transferred to the mobile device. Our measurements (not shown here, to save space) indicate that all of these steps combined require less than one second for all the applications studied. Post-departure cleanup of state at the server and on the mobile device take a few additional seconds.

5 Related Work

Although no previous work addresses the specific combination of concerns addressed by Kimberley, there has been much recent work on building mobile computing systems that take advantage of infrastructure. Want et al [11] focus on the problem of dynamically composing hardware components to yield a computing device with desired capabilities. Balan et al [2] focus on the problem of creating mobile computing applications that can use cyber foraging. Goyal and Carter [5] discuss secure cyber foraging using services provided by VMs in the infrastructure; their work assumes that the VMs have been pre-configured for the desired services. Outside mobile computing, a recent commercial product *vizioncore vPackager* uses VM differencing to support a community of collaborative software developers [10]. Although developed for a completely different purpose and independently of our work, their use of VM differences is superficially similar to our use of VM overlays.

6 Conclusion

Kimberley is the first system to apply VM technology to the problem of provisioning infrastructure for use by mobile computing devices. A good solution to this problem needs to respect the resource limitations of mobile devices while also keeping the configuration complexity of infrastructure low. Kimberley accomplishes the first goal by avoiding VM execution on the mobile device (hence reducing CPU and memory demand) and using indirection to deliver VM overlays to the infrastructure from third-party sites (hence reducing wireless bandwidth demand and energy use on the mobile device). The second goal is accomplished through the use of VM technology. Rather than infrastructure having to be configured for many different applications, it only has to be configured to support Kimberley. The rest of the configuration happens dynamically, through VM overlays. By simplifying configuration, Kimberley lowers a key barrier to the widespread deployment of mobile computing infrastructure.

References

[1] BALAN, R., FLINN, J., SATYANARAYANAN, M., SINNAMOHIDEEN, S., AND YANG, H. The Case For Cyber Foraging. In *Proceedings of the 10th ACM SIGOPS European Workshop* (Saint-Emilion, France, September 2002).

[2] BALAN, R., GERGLE, D., SATYANARAYANAN, M., , AND HERBSLEB, J. Simplifying Cyber Foraging for Mobile Devices. In *Proceedings of the 5th International Conference on Mobile Computing Systems, Applications and Services (MobiSys 2007)* (San Juan, Puerto Rico, June 2007).

[3] CARNEGIE MELLON UNIVERSITY. Diamond: Interactive Search of Non-Indexed Data, 2006-2008. [Online; accessed 20-April-2008 at `http://diamond.cs.cmu.edu`).

[4] FLINN, J., SINNAMOHIDEEN, S., TOLIA, N., AND SATYANARYANAN, M. Data Staging on Untrusted Surrogates. In *Proceedings of the FAST '03 Conference on File and Storage Technologies* (San Francisco, CA, March 2003).

[5] GOYAL, S., AND CARTER, J. A Lightweight Secure Cyber Foraging Infrastructure for Resource-Constrained Devices. In *Proceedings of the Sixth Workshop on Mobile Computing Systems and Applications* (English Lake District, UK, December 2004).

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

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

[8] SATYANARAYANAN, M., GILBERT, B., TOUPS, M., TOLIA, N., SURIE, A., OHALLARON, D., WOLBACH, A., HARKES, J., PERRIG, A., FARBER, D. J., KOZUCH, M., HELFRICH, C., NATH, P., AND LAGAR-CAVILLA, A. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing 11*, 2 (March/April 2007).

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

[10] VIZIONCORE. vpackager version 1.0 user manual, 2007. [Online; accessed on 13-April-2008 at `http://www.vizioncore.com/vPackager.html`].

[11] WANT, R., PERING, T., SUD, S., AND ROSARIO, B. Dynamic Composable Computing. In *Proceedings of the Ninth Workshop on Mobile Computing Systems and Principles (HotMobile 2008)* (Napa, CA, February 2008).

[12] WIKIPEDIA. Lempel-Ziv-Markov chain algorithm — Wikipedia, The Free Encyclopedia, 2008. [Online; accessed 22-April-2008 at `http://en.wikipedia.org/w/index.php?title=Lempel-Ziv-Markov_chain_algorithm&oldid=206469040`].

[13] WIKIPEDIA. VirtualBox — Wikipedia, The Free Encyclopedia, 2008. [Online; accessed 21-April-2008 at `http://en.wikipedia.org/w/index.php?title=VirtualBox&oldid=206252157`].