# Cloud Offload in Hostile Environments

Kiryong Ha, Grace Lewis[†],
Soumya Simanta[†], Mahadev Satyanarayanan

December 2011
CMU-CS-11-146

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]Software Engineering Institute



This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0833882 and IIS-1065336, by an Intel Science and Technology Center grant, and by the Department of Defense (DoD) under Contract No. FA8721-05-C-0003 for the operation of the Software Engineering Institute (SEI), a federally funded research and development center. 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, Intel, DoD, SEI, or Carnegie Mellon University. This material has been approved for public release and unlimited distribution except as restricted by copyright.

**Keywords:** mobile computing, cloud computing, cyber foraging, smartphones, virtual machines, system architecture, cloudlets, disconnected operation, fault tolerance, failure resiliency, denial of service, wireless networks, jamming, ad hoc networks, dynamic VM synthesis, demand-paging, prefetching, cloudlets, battery life, energy efficiency, military operations

## Abstract

Selective offloading of resource-intensive execution from a mobile device to the cloud can extend battery life and broaden the range of supported applications. Unfortunately, the success of this approach is critically dependent on a reliable end-to-end network. This dependence is a serious vulnerability in hostile environments, especially those involving wireless components in their long-haul network segments. We describe an architectural approach to reducing this vulnerability. Using a hierarchical cloud structure, offload is performed on stateless elements that are typically one wireless hop away from a mobile device. The proximity of offload reduces synchronous dependence on vulnerable long-haul network segments. We present experimental results that reflect on key design tradeoffs from a prototype implementation of this architecture.

# 1 Introduction

Offloading resource-intensive computation from a mobile device to the cloud in order to extend battery life or to speed up execution has been the subject of many recent papers [12, 13, 17, 32]. These papers are rooted in work stretching back over a decade on the theme of *cyber foraging* [5, 6, 7, 18, 21, 22, 25, 29, 44, 51]. Commercial applications that use cloud offload now exist: Apple's *Siri* for speech recognition on the iPhone is a good example [4]. An ongoing convergence of mobile computing and cloud computing is clearly under way.

Implicit in this convergence is the assumption that the cloud is easily accessible. In other words, there is good end-to-end network quality, and there are few network or cloud failures to disrupt offloading. While this a reasonable assumption in most of today's use cases, it is untenable in several important contexts that we collectively refer to as *hostile environments*. Foremost among these are theaters of military operations. Another example is an area where disaster recovery is in progress. Even the public Internet may become a hostile environment under conditions of cyber attack. The recent day-long outage of Siri [48, 55] offered a foretaste of the inconvenience and frustration that will be experienced by mobile users when a cloud offload service becomes unavailable.

The U.S. Department of Defense (DoD) has long been a proponent of the use of mobile computing by foot soldiers, with prototype systems such as *Land Warrior* dating back to the mid-1990s [70]. Issues of battery life and the weight of spare batteries have been a nagging concern for the military for many years [15, 16]. Every gram of battery weight subtracts from the ammunition or body armor that a foot soldier can carry. At the same time, the potential mission assistance from resource-intensive technologies such as speech recognition, natural language translation, and face recognition on mobile devices is simply too high to ignore. Reducing energy usage by offloading to the cloud would be an attractive design choice. Unfortunately, the network connection to a distant cloud is vulnerable to wireless jamming or other modes of denial of service (DoS).

After a catastrophic event such as an earthquake, tsunami, hurricane, or terrorist attack, Internet access to the cloud may be compromised for days or weeks. Although limited Internet connectivity may be re-established soon, there will be very high demand on this scarce resource. Yet, within the disaster area, the ability to use resource-intensive applications on mobile devices would be invaluable.

Under conditions of cyber attack, normally well-connected regions of the Internet may be denied access to cloud services. The volume of cyber attacks in the past few years confirms that this is not just a hypothetical possibility [1, 9, 50]. There is growing concern that cyber attacks may soon become major weapons of organized crime as well as instruments of national policy [3, 8, 11]. This puts all of cloud computing at risk, including cloud offload of mobile devices. If the direst of these predictions comes true, we may have no choice but to view the entire wide-area Internet as a hostile environment in the future. Hence, although we focus on military settings in this paper, its results may be of much broader relevance in the future.

# 2 Background

## 2.1 Hungry Mobile Applications

Beyond today's familiar desktop, laptop and smartphone applications is a new genre of software to seamlessly augment human perception and cognition. Consider Watson, IBM's question-answering technology that publicly demonstrated its prowess in 2011 [64]. Imagine such a tool being available anywhere and anytime to rapidly respond to urgent questions posed by an attention-challenged mobile user under stressful conditions such as combat. Such a vision boggles the mind today, but it may be within reach in the next decade. Free-form speech recognition, natural language translation, face recognition, object recognition, dynamic action interpretation from video, and body language interpretation are other examples of this genre of futuristic applications. Combined, they offer enormous potential to enhance the situational awareness of a mobile user and to improve the speed and quality of his responses.

At first glance, it may appear that today's smartphones are already powerful enough to support this genre of applications without need for cloud offload. Android has supported built-in face detection functionality for some time now. The APIs have been extended in Android 4.0 to support tracking of multiple faces and to give detailed information about the location of eyes and mouth [46]. Google's "Voice Actions for Android" performs voice recognition to allow hands-free control of a smartphone [24]. Many computer vision applications that run on resource-limited mobile devices are described in the survey by Lowe [37].

However, upon closer examination, the situation is much more complex and subtle. Consider computer vision, for example. Its computational requirements vary drastically depending on the operational conditions. For example, it is possible to develop (near) frame-rate object recognition (including face recognition [47]) operating on mobile computers *if* we assume restricted operational conditions such as a small number of models (*e.g.*, small number of identities for person recognition), and limited variability in observation conditions (*e.g.*, frontal faces only). The computational demands rapidly outstrip the capabilities of mobile computers as the generality of the problem formulation increases. For example, just two simple changes make a huge difference: increasing the number of possible faces from just a few close acquaintances to the entire set of people known to have entered a building, and reducing the constraints on the observation conditions by allowing faces to be at arbitrary viewpoints from the observer.

Similar tradeoffs apply across the board to virtually all applications of this genre, not only in terms of computational demands but also in terms of dataset sizes. In continuous use under the widest possible range of operating conditions, providing near real-time responses, and tuned for very low error rates, these applications have ravenous appetites for processing, memory and energy resources. Cloud offload is the only hope for meeting this resource demand on a lightweight mobile computer with extended battery life.

## 2.2 Example Use Cases

How could the kinds of applications described in the previous section help a mobile user in a hostile environment? We give a few example use cases below. While the specific scenarios are hypothetical, they are representative of capabilities that the DoD seeks.

*A group of soldiers has just captured a person and needs to confirm his identity. His picture is taken and compared with a continuously updated remote image database. Based on a match with a key enemy officer, the captive is sent to an interrogation center.*
(Source: although the details remain classified, a CNN video interview [14] reports that face recognition technology played a pivotal role in helping Navy SEALs establish positive verification of Osama bin Laden's identity prior to taking action in Abbottabad, Pakistan.)

*A forward unit has been alerted to possible chemical or biological attack. A soldier takes an air sample using a portable sensor that includes a mass spectrometer. Compute-intensive analysis of its output reveals the presence of a known chemical agent. While its concentration is still below the threshold of hazard, a repeat measurement indicates rising concentration. With timely warning, the unit evacuates to safety.*
(Source: Sullivan et al [63] give a good overview of field detection of chemical and biological agents. Examples of commercial portable mass spectrometers can be found at the vendor web site of KD Analytical [28].)

*A soldier is trying to gather information from residents of a remote area that was recently under attack. The soldier knows that these residents speak Pashto rather than Dari or Arabic. With the appropriate settings on his smartphone, the soldier hears translated English from Pashto. His responses generate real-time translations in spoken Pashto.*
(Source: Rattner [49] describes current efforts on language translation in the field in Afghanistan.)

Non-military hostile environments can also benefit from resource-intensive mobile applications. Consider the following disaster recovery scenario: *After the destruction from a massive 9.1 earthquake and resulting tsunami, disaster relief is painfully slow. First responders are guided by now-obsolete maps, surveys, photographs, and building floor plans. Major highways on their maps are no longer usable. Bridges, buildings, and landmarks have collapsed.*

*In desperation, the rescue effort turns to an emerging technology: camera-based GigaPan sensing. Using off-the-shelf consumer-grade cameras in smartphones, local citizens take hundreds of close-up images of disaster scenes. These crowd-sourced images are stitched together into a zoomable panorama using compute-intensive vision algorithms. As new maps and topographical overlays are constructed, rescue efforts speed up and become more effective.*
(Source: The article by Frenkel [23] gives a good overview of GigaPan applications.)

## 2.3 Hostile Environments

*Short-term large-magnitude uncertainty* is the dominant attribute of hostile environments. This contrasts with the well-conditioned, low-uncertainty environment that is experienced by most Internet users today. Note that minor failures and some burstiness of resource demands are already factored into the design of today's Internet applications. For example, TCP retransmission and adaptive windowing masks packet loss and network congestion. Elastic computing mechanisms within a cloud dynamically allocate virtual and physical machine resources based on current workload demands. RAID storage masks unpredictable disk failures and permits online repair. These mechanisms were conceived for benign Internet environments in which failures and overloads are random events rather than the deliberate actions of clever adversaries. By definition, a hostile environment is one that overwhelms engineering practices that are adequate for coping with everyday

uncertainties.

It is important to note that a hostile environment may have extended periods during which everything appears normal. For example, an adversary may remain quiet to induce a false sense of security, and then disrupt network communications at a critical moment. Design for a hostile environment therefore has to be based on worst-case assumptions rather than the average-case assumptions that drive the design choices and economic models of typical Internet applications.

Threats and defenses coevolve continuously in a hostile environment. For example, cryptographic mechanisms that are adequate today may be easily broken tomorrow. Conversely, major threats today may have adequate defenses tomorrow. We assume that current best practices in cyber defenses are rapidly adopted and rigorously enforced at all times. For example, we assume that well-known techniques for end-to-end security are incorporated into mobile clients and into cloud services. These include the use of secure communication tunnels based on end-to-end encryption (using a mechanism such as `ipsec`) and robust mutual authentication technology. We also assume that the strength of these defenses (such as encryption algorithms and key lengths) are regularly upgraded to reflect current best practices. Finally, we assume that physical capture of a mobile device or of the cloud computing infrastructure is rapidly detected. Destruction of these end points is, of course, also a real possibility in combat. With these assumptions in place, we can be confident of the integrity and privacy of computations that are offloaded from mobile devices. The main vulnerability that remains is DoS attacks through disruption of network communication.

Unfortunately, the DoS threat cannot be completely eliminated when most communication is through wireless channels. Jamming of wireless signals continues to be a threat today, in spite of mechanisms such as spread-spectrum transmission, frequency-hopping and other defenses. Recently, "surgical" jamming of wireless signals has been explored as a defense against triggering of improvised explosive devices (IEDs) [68]. This leads to the possibility of inadvertent jamming of communication for cloud offload.

Figure 1 illustrates typical wireless communication links in a modern battlespace [56]. While the operational characteristics of many wireless technologies remain classified, certain broad principles can be identified from the viewpoint of DoS attacks. The wide-area links based on satellite and air support are the most vulnerable to wireless jamming attacks from a distance. The time to repair these links after asset destruction is large, relative to a typical cloud offload operation. It may take many days to launch a replacement satellite or to move an existing one into a new orbit. Time scales of tens of minutes to a few hours are more typical for replacement of lost air assets.

At the tactical level, wireless communication is typically through ad hoc multi-hop networks, as shown in Figure 1. In addition to jamming, these are also vulnerable to unique routing-based attacks [31]. Examples include *wormholes,* in which two rogue nodes create a link with artificially good performance and then drop packets once they are adopted as a good route, and *rushing,* in which an attacker fabricates route requests that result in the network being unable to find routes longer than two hops. Depending on the wireless technology, hop distance can vary from a meter or less to a few tens of kilometers. As a broad generalization, wireless technologies with short range tend to support higher bandwidths, are less vulnerable to jamming from a distance, are less detectable from a distance, and consume less power.

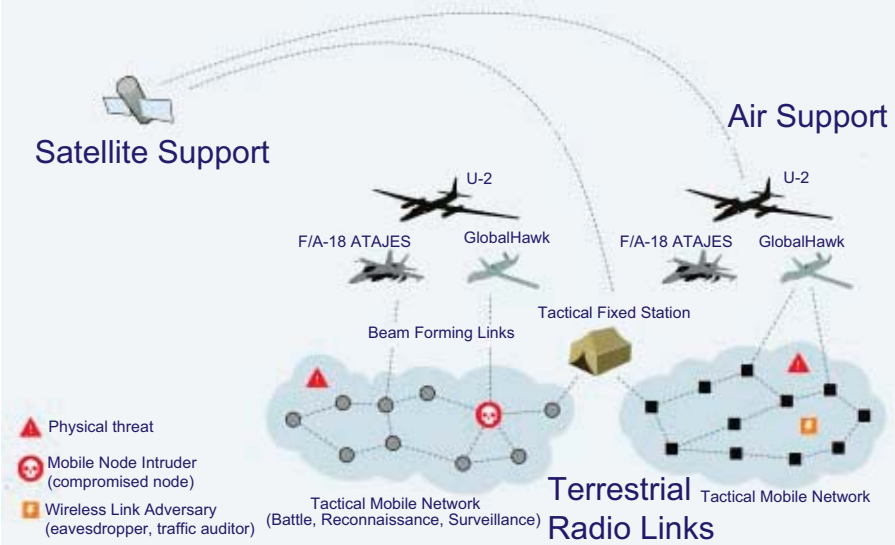


Figure 1: Example Combat Network (Source: [56])

# 3 Architecture and Prototype

Can a cloud offload mechanism be designed to withstand DoS attacks? We assume, as mentioned in Section 2.3, that due diligence is applied at all times to incorporate current best practices in link-level and end-to-end security measures. These can ensure the integrity and privacy of cloud offload operations, but they cannot protect against all DoS attacks. Jamming of long-haul wireless networks, for example, is an ever-present threat in hostile environments. At a mission-critical moment, the cloud may be inaccessible for offload.

There is no silver bullet for this problem: cloud offload is inherently vulnerable to DoS attacks. What is achievable is *a system design that imposes a high work factor [57] for a successful DoS attack.* We describe such an architecture in the following sections. We have implemented a proof-of-concept prototype of this architecture, and include its salient aspects below.

## 3.1 Proximity

*Range* and *directionality* of wireless transmissions are the primary levers of control available to a mobile device in trying to reduce its DoS attack surface. Physical layer mechanisms such as frequency hopping and spread-spectrum transmissions are also relevant. Wise choice of these parameters can greatly increase the work factor needed for a successful DoS attack.

If the offload site can be located very close to the mobile device (ideally one wireless hop away) and ultra-short-range wireless technology is used, then a very high work factor is needed for a successful DoS attack. In that case, jamming attacks from a distance will fail; only attacks from jamming sources that are physically close to the mobile device continue to be threats. If an area larger than the jamming radius can be physically secured around the mobile device, jamming will no longer be a threat. Directional transmissions can further shrink this area. By using only a single wireless hop, threats unique to multi-hop networks can also be avoided. In other words,

placing the offload site close to its mobile device transforms the difficult and poorly-understood problem of DoS attacks into the more tractable and well-understood problem of physical security. In the limit, the work factor for a successful DoS attack is increased to that of physical capture of the mobile device.

An unplanned benefit of placing offload sites close to mobile devices is that it reduces leakage of information for traffic analysis. Inferring an imminent operation merely from recent changes in wireless traffic volumes and traffic patterns (even without access to data content) is a capability of long-standing military importance. Restricting the range of end-to-end offload communication denies that traffic information to distant snoopers.

## 3.2 Hierarchy

This line of reasoning in favor of proximity runs counter to the current ethos of cloud computing. Cloud infrastructure is typically located far from the end-points it serves. For example, Amazon's world-wide public EC2 infrastructure is concentrated in just six sites: US East (N. Virginia), US West (Oregon and N. California), EU (Ireland), Asia Pacific (Singapore and Tokyo). If one of these sites had to be reached from a mobile device located in a forward area (i.e., combat zone) of Figure 1, its end-to-end path would include many network segments that are vulnerable to DoS attacks. More generally, a device's DoS attack surface is greatly increased when its cloud offload site is located far away. Another way to look at this is that increasing the offload distance decreases the work factor for successful DoS attacks.

What accounts for this large difference in strategies between cloud computing today (centralization) and cloud offload in hostile networks (decentralization)? The answer lies in the very different design requirements in the two cases. DoS attacks are considered a rare hazard in cloud computing today. Of far greater importance is the need to reduce the *total cost of ownership* of computing infrastructure by reducing the complexity of system administration. This is achieved by centralization, which provides economies of scale in system administration. In contrast, survivability and high availability of offload sites are the attributes of greatest concern to mobile devices in hostile environments. Decentralization strengthens these attributes, as discussed in the study by Anderson et al [2].

Of course, things are not quite so black and white. Some day, DoS attacks may grow in frequency and impact to the point where cloud computing has to rethink its strategy. The decentralized approach advocated by this paper may then become relevant to cloud computing at large. Conversely, the complexity of system administration cannot be ignored in hostile environments. An approach that can simplify system administration while preserving high availability would be highly desirable. This can be achieved through a system design that appears centralized for system administration, but decentralized for cloud offload.

Figure 2 illustrates such an architecture in which cloud offload infrastructure is organized as a two-level hierarchy. At the heart of this architecture is a large centralized core which is the focus of system administration. This could be implemented as one of Amazon's data centers, and is located in a stable and secure environment, far from physical threats. At the edges of this architecture are offload elements for mobile devices. These elements are dispersed and each is located close to the mobile devices that it serves. Mapping Figure 2 to Figure 1, the offload elements would be located

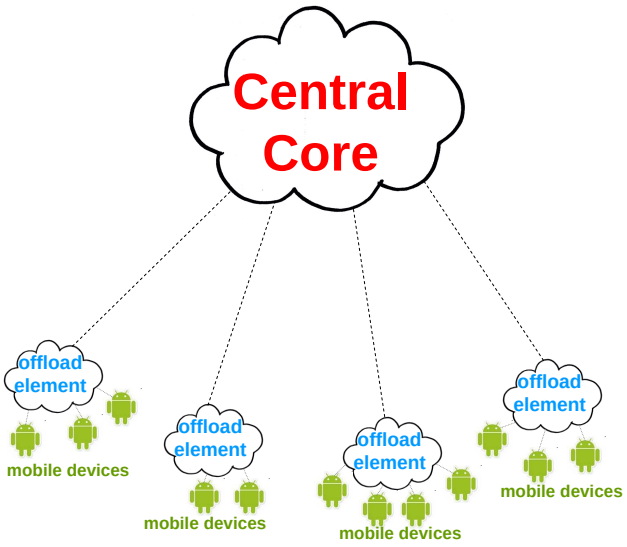


Figure 2: Hierarchical Cloud Offload Infrastructure

at the edges of the tactical mobile networks at the bottom right; the central core would be located far to the top left, reachable only via the satellite link.

A key attribute of this architecture is that the offload elements are *stateless*. They may cache state persistently on their local storage, but the provisioning of the cache is done from the central core. Adding a new offload element or replacing an existing one involves very little setup or configuration effort. After installation, an offload element self-provisions from the central core. A final step of provisioning can occur just before use by a mobile device. Section 3.3 discusses mechanisms for rapid and precise provisioning of offload elements.

This architecture is resilient to wide-area network failures. A mobile device does not need to communicate with the core during an offload operation; it only needs to communicate with its offload element. The mobile device can continue offloading operations even when its offload element is totally disconnected from the core. Communication with the core is only needed during provisioning. Once an offload element is provisioned, its connection to the core can be disrupted without affecting offload service to mobile clients. Further, some of the mechanisms described in Section 3.3 support provisioning even when the core is inaccessible.

Physical motion of a mobile device may take it much closer to an offload element different from the one with which it is currently associated. In that case, a mechanism similar to wireless access point handoff can be executed. A mobile device can thus travel along the periphery of the network, triggering handoffs but never needing to contact the core.

The two-level architecture of Figure 2 bears some resemblance to the concept of a *hybrid cloud* that is gaining traction in enterprises today. A hybrid cloud is a combination of a *public cloud* (such as Amazon's EC2) and a *private cloud* that may use the same technology as the public cloud but is located within the enterprise and solely dedicated to it. Highly sensitive data is only stored on the private cloud. The public cloud is used for elastic compute capacity and for storing less sensitive data. In our architecture, the central core plays the same architectural role as a public

(a) Weather Resistant (b) Solar Powered

Figure 3: Micro Data Centers (Source: [41])

cloud in the hybrid model. However, our offload elements are deployed in the field and are very different from private clouds in both implementation and usage model. A private cloud is a full-fledged data center and its contents are primal, not merely a cache of a public cloud. Closer to our view of offload elements is the concept of a *micro data center* created by Myoonet for developing countries [41]. Figure 3 shows two examples of micro data centers.

Closest in spirit to our view of offload elements is the concept of a *cloudlet* that has been proposed for offloading latency-sensitive applications [53]. Our motivation for a two-level architecture is, of course, completely different: DoS attacks are a threat to all applications, whether latency-sensitive or not. In spite of this fundamental difference, an important point of similarity between cloudlets and the offload elements of Figure 2 is that both are stateless: any persistent state on them is merely a cache of cloud state.

For brevity and ease of exposition, we use the terms "cloudlet" and "cloud" rather than "offload element" and "central core" in the rest of this paper. Keep in mind however, that our use of "cloudlet" is broader and subsumes the original motivation for this architectural element. A cloudlet may be implemented in many forms, with possibilities ranging from the micro data centers of Figure 3 to small self-contained appliances that are installed in motorized elements of an infantry unit. It is not the form factor but the architectural role and statelessness of a cloudlet that is its defining feature.

In theory, the two-level architecture of Figure 2 could be extended to multiple levels. The intermediate nodes of such a hierarchical system could be stateless or stateful, depending on their position in the hierarchy and the role they play. However, while this makes for a more complete and satisfying conceptual picture, we do not see a practical need for more than two levels at present.

## 3.3 Rapid Provisioning

### 3.3.1 Use of Virtual Machines

Optimal partitioning of an application for offload is the subject of ongoing investigation by many researchers. At one extreme is thin client execution in which the entire application is offloaded and only the GUI is retained on the mobile device. At the other extreme is executing the entire

application on the mobile device (i.e., no offload). In between these extremes are many possible ways to partition an application, based on runtime conditions, data content, and application characteristics. Tools and techniques to help with this partitioning span a wide range. Some are language-specific [17], while others require operating system support or other low-level system support [5, 6, 7, 12, 13, 21, 22, 32, 42]. Process models may also vary widely, ranging from a single process to a collection of processes (e.g., for a MapReduce offload). At any given time, many mobile devices may be using a cloudlet. Sometimes a single mobile device may have multiple concurrent applications offloaded on the same cloudlet. The offload support for some applications may require a specific Linux environment, while others require a specific Windows environment. No single "grand unified approach" for cloud offload exists today, and none is likely to emerge in the foreseeable future. So the ability to support offload via a wide range of operating systems, programming languages, and process models is essential. In addition, good isolation between offloaded executions is advisable for safety and robustness.

How should the software environment on a cloudlet be organized to support this full range of possibilities today, and to remain open-ended for future innovations? In many ways, this problem closely resembles the challenge faced by a cloud provider like Amazon who wishes to service the widest possible range of customers with minimal constraints and good isolation. We believe that the same solution, *hardware virtual machines (VMs),* will also work in our context. A separate VM encapsulates the offloaded execution of each mobile device in our prototype. This could be extended to allow multiple VMs per mobile device using a rapid cloning mechanism such as that described by Lagar-Cavilla et al [36]. It would also be relatively simple to support a distinct VM for each application from a mobile device.

The use of VMs on cloudlets enables clean separation of concerns. The complex and messy problem of dynamically configuring software on a cloudlet to service a mobile device is avoided. Instead, the problem is transformed into the simpler problem of rapidly delivering a precisely preconfigured VM to the cloudlet. A VM cleanly encapsulates and separates a transient *guest* software environment from the permanent *host* software environment of the cloudlet infrastructure. The interface between the host and guest environments is narrow, stable, and ubiquitous. This ensures the longevity of cloudlet investments and greatly increases the chances of compatibility between a mobile device and a cloudlet. The malleable software interfaces of the offloaded components of a mobile application are encapsulated within the guest environment. As a result, a VM-based approach is less brittle than alternatives such as process migration or software virtualization. It is also less restrictive and more general than language-based virtualization approaches that require applications to be written in a specific language such as Java or C#.

Our prototype uses KVM, a Type 2 VMM, embedded in an Ubuntu Linux host. This choice was made primarily for ease of implementation. The rich host environment simplifies implementation of supporting software for functionality such as resilient data access (Section 3.4) and cloudlet discovery (Section 3.5). A production-quality implementation of a cloudlet would likely use a Type 1 VMM such as Xen or VMware ESX. In that case, supporting functionality would likely be implemented as VMs themselves.

### 3.3.2 VM Delivery Strategies

We envision the guest contents of an offload VM being created and tested offline. An instance of that VM is then booted up and brought to a state where it is ready to service requests from a mobile device. At that point it is suspended. The suspended VM instance typically has distinct representations of its persistent state (i.e., its virtual disks) and its volatile state (i.e. memory, registers, etc.). We refer to these as the *disk image* and *memory snapshot* respectively, and to the combination as the *VM image.* If the VM instance is shut down rather than suspended, there is no memory snapshot saved. Resuming such a VM instance is equivalent to powering up hardware, resulting in a fresh boot of the guest operating system. Since startup delay is an important performance metric when using cloud offload, we expect that most VM instances will be resumed rather than rebooted. A cold boot may be needed in rare instances, such as when a unique local hardware device on a cloudlet has to be detected and configured into the guest OS.

A typical VM image is many gigabytes in size, possibly tens of gigabytes. In the context of a hostile network, efficient dissemination of VM images to cloudlets is a major challenge. In the rest of this section, we examine a range of approaches that together define a complex tradeoff space. We quantify these tradeoffs through experimental results in Section 4. Once a cloudlet has acquired a copy of a VM, it can treat it as a persistent cache copy and retain it until the space has to be reclaimed. Persistent caching of VMs takes advantage of temporal locality (the same mobile device may be associated with the cloudlet again in the future) as well as communal locality (many mobile devices may use the same VM image for offload).

**Bulk Transfer from Cloud** $(B_{cloud})$**:** The simplest way to deliver a VM image to a cloudlet is to download it in its entirety from the cloud. If the end-to-end bandwidth between cloudlet and cloud approaches 10 Mbps (typical for well-connected Internet sites today), the transfer of a 10 GB VM will take over two hours. This is clearly unacceptable in time-critical situations, but it may be acceptable for routine software upgrades of cloudlets because they can be done in the background. DoS attacks may disrupt the transfer many times, but eventual completion is possible if standard mechanisms to continue transfers after interruption are used. Mechanisms such as `rsync` may be useful if the VM to be transferred has significant content similar to an existing VM at a cloudlet. More sophisticated deduplication mechanisms [65, 66, 67] may further reduce the number of bytes transferred. No energy is consumed on the mobile device with this approach. Upon successful completion, the entire VM image is available on the cloudlet.

**Bulk Transfer from Mobile** $(B_{mobile})$**:** A mobile device may have adequate local storage to hold copies of all the critical VMs that it needs for offload. If it associates with a cloudlet that is missing a VM, the mobile device can directly transfer it over the one-hop wireless link. This is likely to be much faster than a cloud-cloudlet transfer. For example, a 10 GB VM will take about 13 minutes at 100 Mbps. Although this is likely to be unacceptably long for a mission-critical situation, it may be one of the few options available if the cloudlet is disconnected due to DoS attacks. The energy drain on the mobile device is likely to be high. A hybrid approach would transfer data from the cloud when possible, but would continue the transfer from the mobile device during periods when the cloudlet is disconnected. This would save energy on the device, yet complete transfer of the VM.

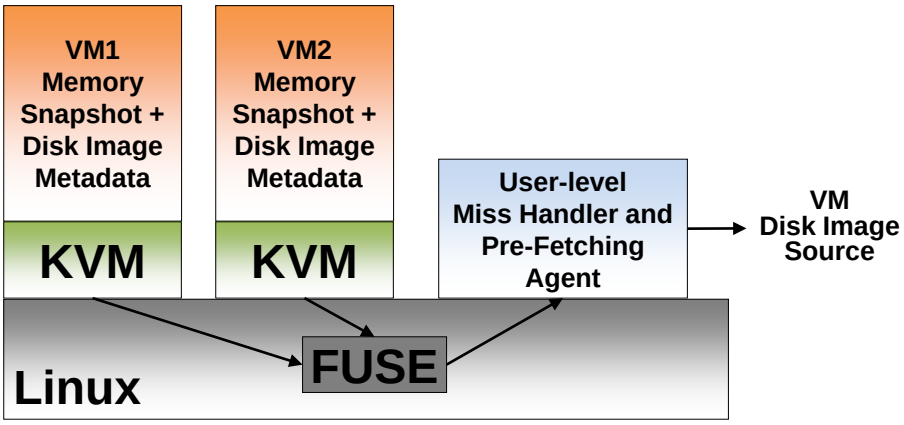


Figure 4: Demand Paging of VM Images on Cloudlets

**Demand-page from Cloud ($D_{cloud}$):** Rather than waiting for an entire VM image to be fetched from the cloud, there may be situations where rapid launch of the VM is desirable. This would only require the initial working set of the VM to be fetched and then the VM could be launched. Missing parts of the VM image could then be fetched on demand as execution proceeds. If bandwidth permits, VM execution could overlap prefetching.

While demand paging of VM images has the benefit of fast VM launch, there are some drawbacks. If the cloud cannot be accessed to service a demand miss, offload execution will stall. Also, transferring a VM image through a streaming bulk transfer is likely to be more efficient than a series of short data transfers in response to demand misses. On the other hand, much of a VM image may never be accessed during execution if it has been sloppily constructed. Demand paging only fetches those parts of a VM image that are actually used.

Figure 4 shows how demand paging of VM images is implemented in our prototype. The VM disk image is mapped to a file in the FUSE file system on the host. Disk I/O from a VM is routed by FUSE to a user-level agent that we have implemented to perform miss handling and prefetching. Standard HTTP `GET` requests are used to service demand misses. A web server in the cloud can service these `GET` requests. A limitation of all current VMMs (including KVM) is that they require the entire memory snapshot to be present before a VM can be resumed. Our experimental results in Section 4 therefore reflect this limitation. We are extending KVM to support demand paging of memory snapshots, just as we currently support demand paging of disk images. When this extension is complete, the user-level agent in Figure 4 will handle requests for missing regions of both memory snapshots and disk images. We are also extending this agent to accept external hints and to use them in prefetching memory and disk images.

**Demand-page from Mobile ($D_{mobile}$):** The implementation of VM demand paging shown in Figure 4 is agnostic regarding where the full VM image is located and which entity handles miss traffic. If the cloud is inaccessible, the mobile device could handle demand misses. This will require sufficient persistent storage on the device to hold all the VMs that are likely to be needed for offload. Note that these VMs are never executed on the mobile device, so this approach can be used even when the device and cloudlet are of different hardware architectures (e.g., ARM mobile

device and x86 cloudlet). A number of web servers are available for Android mobile devices. Our prototype uses `kWS`.

Demand paging from the mobile device is likely to be faster than demand paging from the cloud because of the higher mobile-cloudlet bandwidth. However, it will shorten battery life on the device. A hybrid approach could treat the mobile device as a failover site for handling demand misses when the cloud is inaccessible. This will lengthen the average delay for servicing a miss, since the cloudlet has to first try the cloud, timeout, and then redirect the miss to the mobile device. On the other hand, device energy use is likely to be small if most misses are handled by the cloud.

**Synthesis from Mobile** ($S_{mobile}$)**:** Any of the four approaches above can be combined with block-level deduplication to take advantage of cached VM state at a cloudlet. This would reduce the volume of data that has to be fetched from the cloud or mobile device. Previous work has shown that block-level deduplication can be effective on VMs [43, 65]. However, the benefit is modest relative to the implementation effort involved. We have therefore developed a higher-level approach to deduplication that is more effective. This approach, called *dynamic VM synthesis,* exploits knowledge of the process by which VMs are typically created.

In this technique, we refer to a VM used for offloading as a *launch VM*. A launch VM is typically created by installing relevant software into a *base VM* in which a minimally-configured guest OS has been installed. There are no constraints on the guest OS; our prototype works with both Linux and Windows. The binary difference between the base VM image and the launch VM image is called a *VM overlay.* We anticipate that a relatively small number of base VMs will be popular on cloudlets at any given time. To increase the chances of success, a mobile device can carry overlays for multiple base VMs and discover the best one to use through negotiation with the cloudlet. As in the case of demand paging, the cloudlet and mobile device can have different hardware architectures: the mobile device is merely serving as transport for the VM overlay.

In our prototype, the overlay is created using the `xdelta` binary differencing tool. Our experience has been that `xdelta` generates smaller overlays than the native VM differencing mechanism provided by KVM. The VM overlay is then compressed using the Lempel-Ziv-Markov algorithm, which is optimized for fast decompression at the price of relatively slow compression [69]. This is an appropriate tradeoff because decompression takes place in the critical path of execution at runtime and contributes to user-perceived delay. Further, compression is only done once offline but decompression occurs each time VM synthesis is performed.

As Figure 5 shows, dynamic VM synthesis reverses the process of overlay creation. A mobile device delivers the VM overlay to a cloudlet that already possesses the base VM from which this overlay was derived. The cloudlet decompresses the overlay and applies it to the base to derive the launch VM. Normally, each offloading session starts with a pristine instance of the launch VM. However, there are some use cases where modified state in the launch VM needs to be preserved for future offloads. For example, the launch VM may incorporate a machine learning model that adapts to a specific user over time. Each offload session then generates training data for an improved model that needs to be incorporated into the VM overlay for future offload sessions. This is achieved in Figure 5 by generating a *VM residue* on the cloudlet that can be sent back to the mobile device and incorporated into its overlay.

Demand paging and VM synthesis are contrasting approaches to efficient delivery of VM state.

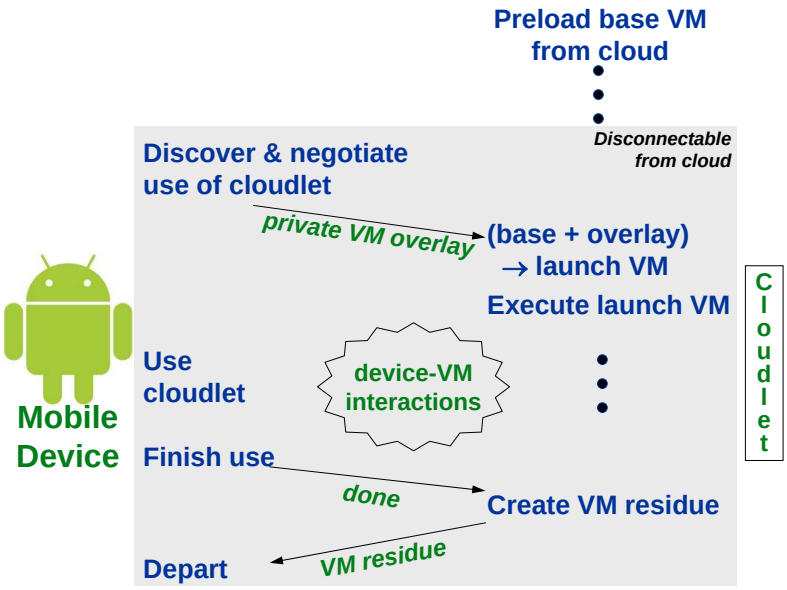


Figure 5: Dynamic VM Synthesis from Mobile Device

Both work even if the cloud is inaccessible. Synthesis requires the base VM to be available on the cloudlet. In contrast, demand paging from mobile works even for a freshly-created VM that has no ancestral state on the cloudlet. Synthesis can use efficient streaming to transmit the overlay, while demand paging incurs the overhead of many small data transfers. However, some of the state that is proactively transferred in an overlay may be wasted if the launch VM includes substantial state that is not accessed. Synthesis incurs a longer startup delay before VM launch. However, once launched, the VM incurs no stalls. This may be of value for offloads of soft real-time mobile applications such as augmented reality.

**Synthesis from Cloud** $(S_{cloud})$**:** Rather than transmitting the overlay, a mobile device can direct the cloudlet to obtain it from the cloud. This reduces energy use on the mobile device, but is vulnerable to cloud-cloudlet network disruptions during overlay transfer. In other respects, the contrast between demand paging and VM synthesis discussed earlier applies here too.

## 3.4 Cloud-Wide Resilient Data Access

A large external corpus of data is an important part of many applications that are relevant to this paper. For example, a face recognition application typically requires a collection of facial images of persons of interest. This collection may grow and shrink over a period of time, as faces of interest are added and removed. These updates may be performed in the field at many different locations. VMs running the face recognition application will need access to relevant parts of this data collection.

These types of applications need cloud-wide distributed data access that is resilient to network disruptions. Our prototype uses a cloud-based distributed file system with a persistent cache of file data on each cloudlet: release 6.9.5 of the Coda File System [10, 52].

As Figure 6 illustrates, the cloud-wide file namespace associated with a cloudlet's Coda cache

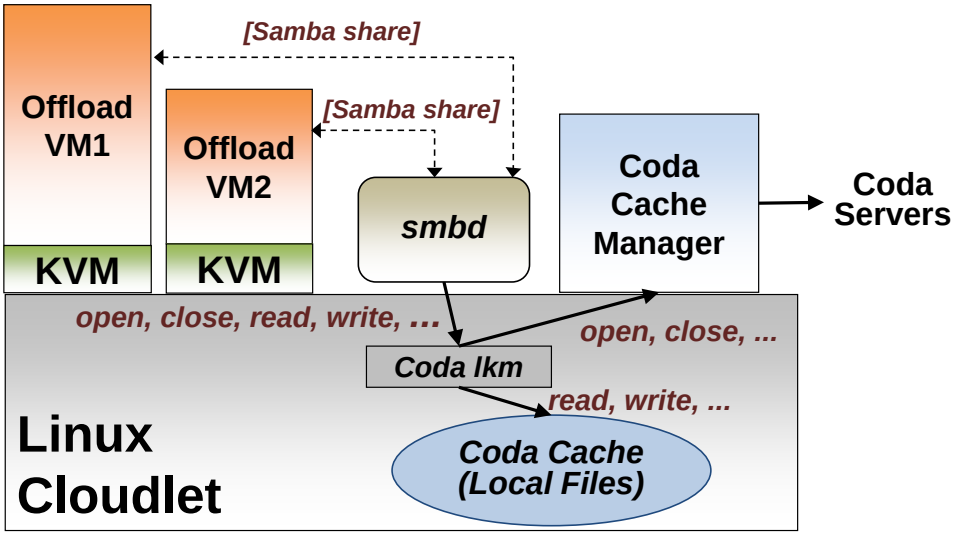


Figure 6: VM Access to Shared Data

is exported as a file share through Samba [19] to each offload VM executing on that cloudlet. Thus, regardless of guest operating system type, offload application code within a VM sees a Samba file share that represents its external data corpus. The Samba daemon, `smbd`, maps this to a region of the cloud-wide shared namespace.

Since Coda has been described extensively in the literature, we only provide a brief summary here. Coda is cloud-centric and its key architectural features are inherited from AFS [26]. These include aggressive client caching on local disks, callback-based cache coherence, location transparency, organization of the file name space into logical volumes to facilitate system administration, and file protection based on access control lists. Some of the specific aspects of Coda that make it well suited for cloudlet use include:

*Frugal Bandwidth Use (efficient logging and adaptive replay) [40]:* Updates are appended to a persistent operation log that is kept as short as possible through log optimizations. Transfers of large files occur as a series of fragments and are resumable after network failures.

*Hoarding (user-guided prefetching for disconnected operation) [30]:* To prepare for unexpected disconnection, Coda uses *hoarding* to ensure that critical objects are cached. This is implemented by combining implicit and explicit sources of information into a priority-based cache management algorithm. Guidance for hoarding can be obtained manually or through third-party tools.

*Intermittent Networks (rapid cache validation) [40]:* Coda clients track server state at two levels of granularity: on individual objects and on entire volumes. This improves the speed of cache revalidation at the cost of precision of invalidation. This is a good tradeoff when network disruption is frequent.

*Update Conflict Resolution [33, 34, 35]:* Coda provides mechanisms to detect and transparently resolve update conflicts on directories and files. For directories, it uses a set of built-in heuristics for resolution. For files, it provides a plugin interface for application-specific resolvers

to be installed and dynamically executed.

*Portable and Minimally-Invasive Implementation [61]:* Most complexity is encapsulated into a user-level cache manager. Only a tiny loadable module for redirecting file system calls resides in the kernel. This kernel module is part of Linux.

Data consistency is clearly an important issue in this context. The implementation shown in Figure 6 offers different levels of consistency within and across cloudlets. Within a cloudlet, near-POSIX semantics apply to all files accessed via Samba shares even if they are in different VMs. This is because all the Samba shares map to a single Coda cache in the host environment of the cloudlet, and that cache is composed of local Linux files. This allows correct functioning of offload operations that, for example, use a collection of VMs on a many-core cloudlet to implement a MapReduce task. Across cloudlets, the consistency offered is the classic Coda consistency guarantee: one-copy semantics at open-close granularity at all connected sites and eventual consistency at all currently-inaccessible sites.

### 3.5 Cloudlet Discovery

The ability to rapidly discover nearby cloudlets, to make an optimal selection, and to securely associate are important capabilities in a cloud offload. There are many well-known service discovery mechanisms such as UPnP, Bluetooth Service Discovery, and Jini. These are typically implemented at Layer-3 of the network stack. A Layer-2 protocol for faster service discovery has been described by Sud et al [62]. Our prototype currently has a minimal implementation of cloudlet discovery based on Avahi, an implementation of ZeroConf. We plan to expand this to a complete implementation of cloudlet discovery in the future.

## 4 Evaluation

To quantify and better understand the VM delivery approaches discussed in Section 3.3.2, we have conducted a number of experiments on our prototype. Since bulk transfer is straightforward, we focus on demand paging and synthesis. We ask two questions in each experiment:

- What is the total time and breakdown to deliver a VM, launch it, and perform an offload operation?
- How does energy consumption on the mobile device differ across approaches?

### 4.1 Applications & Experimental Setup

We offloaded four applications on a cloudlet:

**OBJECT:** Linux C++ application based on the CMU MOPED object recognition libraries [39]. It returns the identities of recognized objects in an input image.

**FACE:** Windows XP C++ application based on the OpenCV library [45]. It returns the coordinates and identities of recognized faces in an input image.

**SPEECH:** Windows XP Java application based on the CMU Sphinx-4 speech recognition toolkit [60]. It returns a text transcription of a WAV input file.

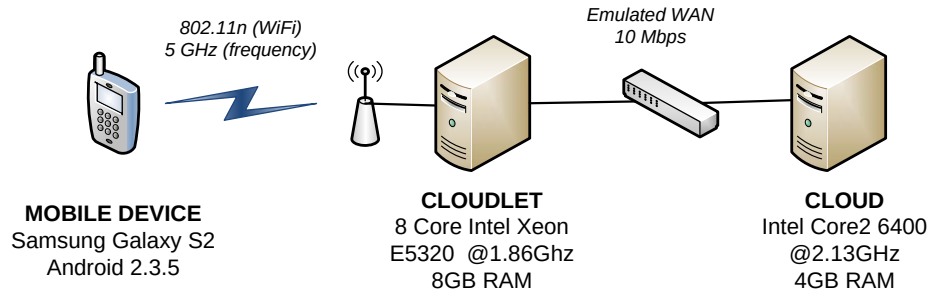


Figure 7: Experimental Setup

**NULL:** Empty application to serve as a baseline.

All experiments were conducted using the configuration shown in Figure 7. The mobile device is a Samsung Galaxy S2 smartphone running Android 2.3.5. It connects to the cloudlet via 802.11n Wi-Fi at 5 GHz. The cloudlet is connected to the cloud via a 1 Gbps Ethernet and a network emulator that can throttle bandwidth to emulate a WAN. We measured energy usage with a Power Monitor from Monsoon Solutions Inc. and the corresponding Power Tool software [38]. To ensure good experimental control, we scripted all interactive inputs. Three runs of each experiment were done. Since observed standard deviations were low (less than 4% for time and less than 5% for energy) they are not explicitly shown in our results.

## 4.2 Benchmarks and Metrics

We established four benchmarks that map to the demand paging and synthesis strategies presented in Section 3.3.2. In each of these cases the VM launch is initiated by the mobile device.

**Demand Page from Cloud** $(D_{cloud})$**:** The cloudlet retrieves the VM memory snapshot and VM disk image metadata from the cloud and launches the VM using only the memory snapshot. The VM disk image remains on the cloud and pages are retrieved on demand.

**Demand Page from Mobile** $(D_{mobile})$**:** Similar to the previous benchmark with the difference that the VM memory snapshot and VM disk image metadata are retrieved from the mobile device and the VM disk image remains on the mobile device.

**Synthesis from Cloud** $(S_{cloud})$**:** The cloudlet contains the base VM disk image and the base VM memory snapshot. It retrieves the VM disk overlay and the VM memory snapshot overlay from the cloud and applies them to the base disk image and base memory snapshot. After that, the cloudlet launches the VM.

**Synthesis from Mobile** $(S_{mobile})$**:** Similar to the previous benchmark with the difference that the VM disk image overlay and VM memory snapshot overlay are retrieved from the mobile device instead of the cloud.

Table 1 and Table 2 show VM information for the demand paging and synthesis cases. The metrics captured during the experiments were end-to-end VM launch time, first run time, and energy consumption on the mobile device. In the demand paging cases, VM launch time is the sum of memory snapshot transfer time, disk image metadata transfer time, snapshot decompression

| App | App Size (MB) | VM Disk Image (GB) | VM Disk Image Metadata (MB) | Compr VM Mem Snapshot (MB) |
|---|---|---|---|---|
| OBJECT | 27.50 | 2.48 | 5.45 | 136 |
| FACE | 17.65 | 1.57 | 5.45 | 121 |
| SPEECH | 51.04 | 1.57 | 5.45 | 189 |
| NULL | 0.00 | 2.48 | 5.45 | 65 |

Table 1: VM Information for Demand Paging Experiments

| App | App Size (MB) | Base Disk Image (GB) | Base Mem Snap (MB) | Compr Disk Image Ovlay (MB) | Compr Mem Snap Ovlay (MB) |
|---|---|---|---|---|---|
| OBJECT | 27.5 | 2.50 | 474.49 | 30.27 | 134.55 |
| FACE | 17.65 | 1.61 | 357.69 | 50.54 | 44.13 |
| SPEECH | 51.04 | 1.61 | 357.69 | 96.60 | 89.23 |
| NULL | 0.00 | 2.50 | 474.49 | 0.01 | 0.12 |

Table 2: VM Information for Synthesis Experiments

time and VM resume time. In the synthesis cases VM launch time is the sum of VM disk image overlay transfer time, VM memory snapshot overlay transfer time, overlay decompression time, VM synthesis time and VM resume time.

## 4.3 Results and Discussion

**End-to-End VM Launch Time:** Figure 8 shows benchmark results for the different VM delivery strategies for each application. The graph shows that data transfer is the largest component of end-to-end VM launch time and that it is greater for the VM delivery strategies from the cloud. This is expected because the bandwidth is smaller for cloud-cloudlet communication (10 Mbps). The mobile device used in our experiments supports 32–38 Mbps. Data transfer times should be even smaller with other shorter-range, higher-bandwidth networks.

The graph also shows the effect of VM image size. The synthesis strategies transfer only the overlays for the VM image which contain the parts that are specific to the application being offloaded. As shown in Tables 1 and 2 the overlays for the most part are much smaller in size compared to the full VM image size. For the FACE application, for example, the disk overlay is 3.2% of the disk image and the memory overlay is 36% of the memory snapshot. Because overlays are smaller, end-to-end VM launch time for synthesis is expected to be smaller than for demand paging even if both the VM image and memory snapshot overlays have to be transferred before the VM can be launched. However, if the application requires a lot of disk space or memory then

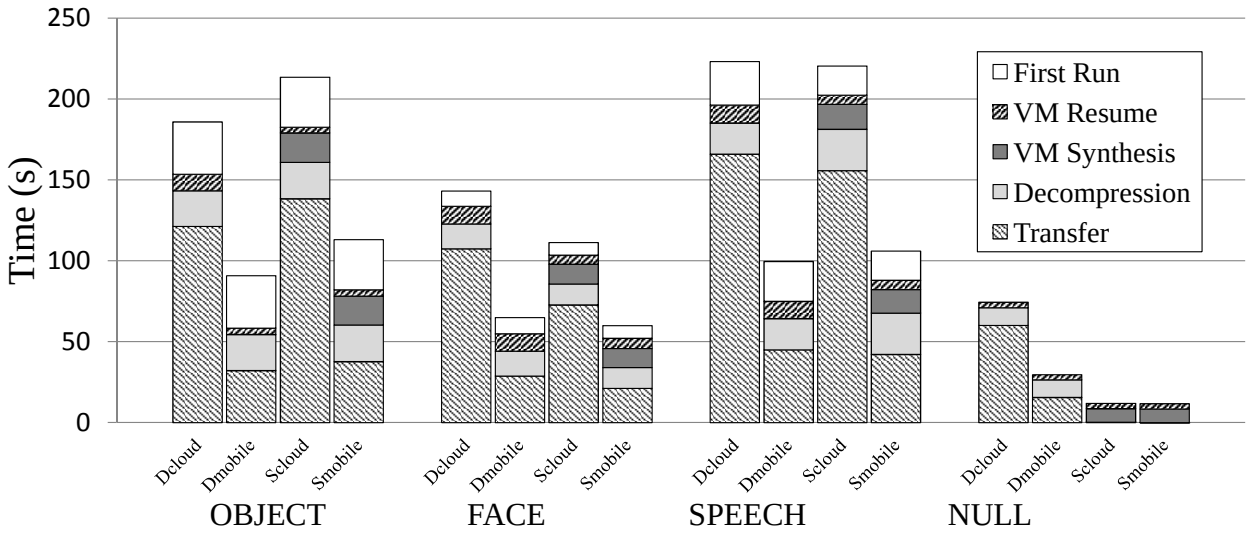


Figure 8: Total Benchmark Time

the sum of the sizes of the two overlays could be greater than the size of the memory snapshot required for demand paging. This is the case with OBJECT, where the disk overlay is 1.2% of the disk image but the memory overlay is 98% of the memory snapshot, which explains the greater end-to-end VM launch times for the synthesis strategies.

**Application Execution Time:** Figure 8 shows that first run time is very similar between the demand paging and synthesis strategies for all applications. The only one that shows a slight difference is SPEECH (demand paging takes 9 more seconds than synthesis). This is contrary to our intuition because we expected demand paging to have greater first run times. The rationale is that although demand paging can generate the launch VM faster because it only requires the VM memory snapshot, it requires disk page fetching during the first run because it has no disk image data. SPEECH is the most data-intensive application because it reads large language model files. This explains why it shows a slight difference in first run times.

To further understand the effects of demand paging on data-intensive applications we modified OBJECT so that it would load the object-modeling files (25MB) upon client request instead of having them pre-loaded in memory, which would make them part of the VM memory snapshot. Figure 9 shows the original SPEECH data along with the results of modified OBJECT. In both cases, we can clearly see that the synthesis strategy has a smaller first run time but a larger first launch time which is consistent with our intuition. This difference does not appear to be significant because data transfer time is still the dominant part of end-to-end VM launch time. However, the effect of demand paging will increase linearly as the size of the files increases over time due to additional training or object-modeling files.

**Energy :** Energy consumption in all the experiments is measured from the moment that the mobile device initiates VM launch until the first run ends. Figure 10 shows that $D_{mobile}$ and $S_{mobile}$ consume more energy than $D_{cloud}$ and $S_{cloud}$ respectively. The average energy savings is 14.3%

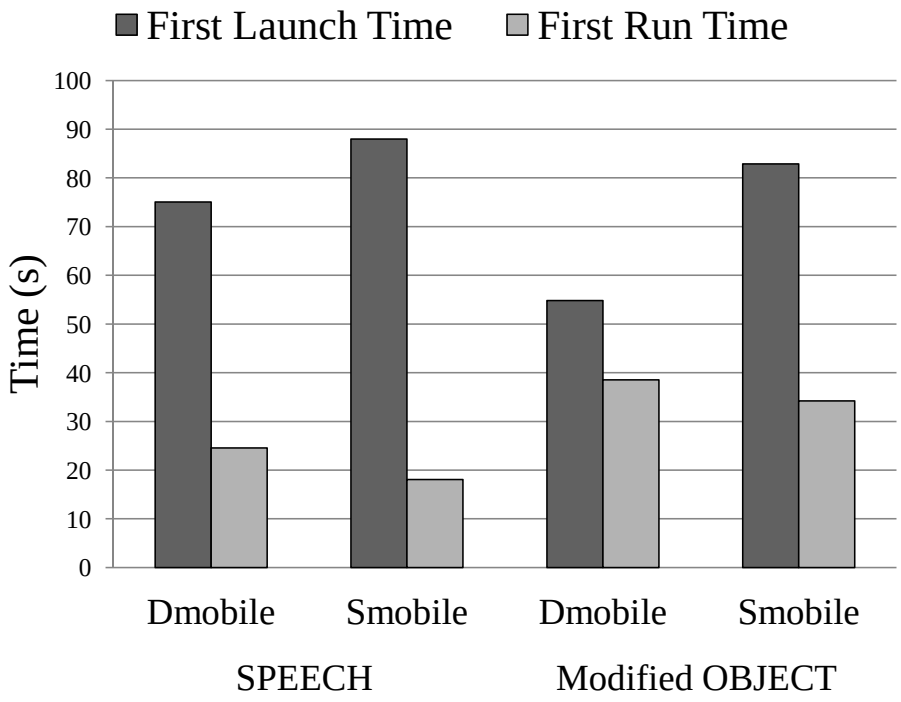


Figure 9: Demand Paging vs. Synthesis

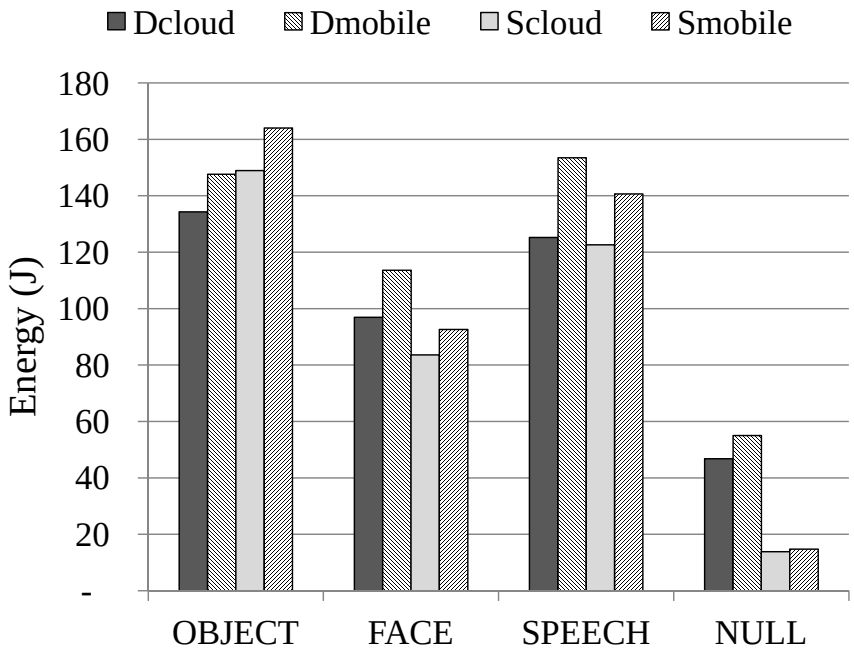


Figure 10: Energy consumption on mobile device

for demand paging cases and 9.6% for synthesis cases. These savings seem small. However, data transfer from the cloud takes approximately twice as long as from the mobile device (see Figure 8) and during this time the mobile device is in the idle state waiting for a response. Our experiments show that average power in the idle state is 430.44 mW, which is significant and contributes to this lower than expected savings.

# 5 Related Work

To the best of our knowledge, this work is the first to investigate the challenges of cloud offload in hostile environments and to propose an architectural solution to the problem. All previous work has assumed that acceptable networking conditions prevail between a mobile device and its offload site. Although the bandwidth and latency of this connectivity may vary, a universal assumption in previous work has been that connectivity is "good enough." The performance impact of network bandwidth on cloud offload, including its impact on application partitioning strategy and energy usage on a mobile device, has been studied by a number of researchers [17, 21, 22, 32]. Previous research has investigated the challenges of hostile environments from many viewpoints other than cloud offload. Examples include access to shared files in the face of network failures [30], supporting delay-tolerant networks [20], detection of DoS attacks [27, 54], and rapid deployment in service-oriented architectures [58]. The use of "data mules" in sensor networks to transport data through physical mobility [59] resembles our use of a mobile device to deliver a VM to a cloudlet.

# 6 Conclusion

Today, there is substantial consensus in the research community that cloud offload of resource-intensive application execution is a core technique in mobile computing. In this paper, we have explored the unique challenges of using this technique in hostile environments. We have described a decentralized two-level architecture that cleanly separates the concerns of offloading execution and precisely provisioning the offload sites. A difficult problem exposed by this architecture is rapid delivery of large VMs to offload sites. We have developed a number of different approaches to accomplish this task, with performance and availability tradeoffs across them. We have implemented a proof-of-concept prototype of our proposed architecture. Experimental results from this prototype quantify the tradeoffs of different VM delivery approaches with respect to VM launch time, application execution time, and energy consumption on the mobile device.

If cyber attacks become more prevalent on the public Internet, cloud offload of mobile devices will become increasingly unreliable. Some day, the entire public Internet may have to viewed as a hostile environment. The issues explored in this paper in the context of military settings will then be of much broader relevance.

# Acknowledgements

We wish to thank many individuals who contributed to the work reported here. Gene Cahill implemented the speech application mentioned in Section 4.1. Alvaro Collet implemented the object recognition application reported in that section. Yoshihisa Abe and Benjamin Gilbert guided us in the implementation of demand paging and prefetching for VMs. Babu Pillai helped us to think through many subtle aspects of dynamic VM synthesis. Ardalan Amiri Sani guided us in the methodology used for the energy measurements of Section 4.3. Edwin Morris championed this work and helped us obtain the resources that we needed for it. Jan Harkes helped us with the characterization of resource-intensive mobile applications in Section 2.1. Benjamin Gilbert, Jan Harkes, Edwin Morris, Wolfgang Richter, and Roxana Geambasu read drafts of this paper and provided us with valuable suggestions for improving its content and presentation.

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0833882 and IIS-1065336, by an Intel Science and Technology Center grant, and by the Department of Defense (DoD) under Contract No. FA8721-05-C-0003 for the operation of the Software Engineering Institute (SEI), a federally funded research and development center. 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, Intel, DoD, SEI, or Carnegie Mellon University. This material has been approved for public release and unlimited distribution except as restricted by copyright.



# References

[1] ANDERSON, N. Massive DDoS attacks target Estonia; Russia accused. *ARS Technica* (May 2007). `http://arstechnica.com/news.ars/post/20070514-massive-ddos-attacks-target\-estonia-russia-accused.html`.

[2] ANDERSON, R. H., FELDMAN, P. M., GERWEHR, S., HOUGHTON, B., MESIC, R., PINDER, J., ROTHENBERG, J., AND CHIESA, J. Securing the U.S. Defense Information Infrastructure. Tech. Rep. MR-993-OSD/NSA/DARPA, Rand Corporation, 1999.

[3] ANDRESS, J., AND WINTERFELD, S. *Cyber Warfare: Techniques, Tactics and Tools for Security Practitioners*. Syngress, 2011.

[4] APPLE. iPhone 4S - Ask Siri to help you get things done. `http://www.apple.com/iphone/features/siri.html`.

[5] 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).

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

[7] BALAN, R., SATYANARAYANAN, M., OKOSHI, T., AND PARK, S. Tactics-based Remote Execution for Mobile Computing. In *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services* (San Francisco, CA, May 2003).

[8] BRENNER, S. *Cyber Threats: The Emerging Fault Lines of the Nation State*. Oxford University Press, 2009.

[9] BRODKIN, J. Government-sponsored cyberattacks on the rise, McAfee says. *NetworkWorld* (November 2007). `http://www.networkworld.com/news/2007/112907-government-cyberattacks.html`.

[10] CARNEGIE MELLON UNIVERSITY. Coda File System, December 2011. `http://coda.cs.cmu.edu`.

[11] CARR, J. *Inside Cyber Warfare: Mapping the Cyber Underworld*. O'Reilly, 2010.

[12] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. CloneCloud: Elastic Execution between Mobile Device and Cloud. In *Proceedings of EuroSys 2011* (Salzburg, Switzerland, April 2011).

[13] CHUN, B.-G., AND MANIATIS, P. Augmented Smart Phone Applications Through Clone Cloud Execution. In *Proceedings of HotOS XII, the 12th Workshop on Hot Topics in Operating Systems* (Monte Verita, Switzerland, May 2009).

[14] CNN. How facial recognition technology works, May 2011. `http://www.ktar.com/category/videos-article/20110504/How-facial-recognition-software-work-(VIDEO)/`.

[15] COMMITTEE ON ELECTRIC POWER FOR THE DISMOUNTED SOLDIER. *Energy-Efficient Technologies for the Dismounted Soldier"*. National Research Council, 1997.

[16] COMMITTEE ON SOLDIER POWER/ENERGY SYSTEMS. *Meeting the Energy Needs of Future Warriors*. National Research Council, 2004.

[17] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., WOLMAN, A., SAROIU, S., CHANDRA, R., AND BAHL, P. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services* (San Francisco, CA, June 2010).

[18] DE LARA, E., WALLACH, D. S., AND ZWAENEPOEL, W. Puppeteer: Component-based Adaptation for Mobile Computing. In *Proceedings of the 3rd Conference on USENIX Symposium on Internet Technologies and Systems* (2001).

[19] ECKSTEIN, R., COLLIER-BROWN, D., AND KELLY, P. *Using Samba*. O'Reilly, Inc., November 1999.

[20] FALL, K. A Delay-Tolerant Network Architecture for Challenged Internets. In *Proceedings of SIGCOMM 2003* (Karlsruhe, Germany, August 2003).

[21] FLINN, J., NARAYANAN, D., AND SATYANARAYANAN, M. Self-Tuned Remote Execution for Pervasive Computing. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems* (Schloss Elmau, Germany, May 2001).

[22] FLINN, J., PARK, S., AND SATYANARAYANAN, M. Balancing Performance, Energy Conservation and Application Quality in Pervasive Computing. In *Proceedings of the 22nd International Conference on Distributed Computing Systems* (Vienna, Austria, July 2002).

[23] FRENKEL, K. A. Panning for Science. *Science 330* (November 2010).

[24] GOOGLE. Voice Actions for Android, 2011. `http://www.google.com/mobile/voice-actions/`.

[25] GOYAL, S., AND CARTER, J. A Lightweight Secure Cyber Foraging Infrastructure for Resource-constrained Devices. In *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications* (2004).

[26] HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems 6*, 1 (February 1988).

[27] HUSSAIN, A., HEIDEMANN, J., AND PAPADOPOULOS, C. A framework for classifying denial of service attacks. In *Proceedings of SIGCOMM 2003* (Karlsruhe, Germany, 2003).

[28] KD ANALYTICAL. Gas Chromatograph / Mass Spectrometry (GC/MS), 2011. `http://www.kdanalytical.com/instruments/technology/gc-ms.aspx`.

[29] KEMP, R., PALMER, N., KIELMANN, T., SEINSTRA, F., DROST, N., MAASSEN, J., AND BAL, H. eyeDentify: Multimedia Cyber Foraging from a Smartphone. In *Proceedings of the 11th IEEE International Symposium on Multimedia* (San Diego, CA, December 2009).

[30] KISTLER, J. J., AND SATYANARAYANAN, M. Disconnected Operation in the Coda File System. *ACM Transactions on Computer Systems 10*, 1 (February 1992).

[31] KONG, J., HONG, X., AND GERLA, M. A New Set of Passive Routing Attacks in Mobile Ad-hoc Networks. In *IEEE Military Communications Conference (MILCOM03)* (Boston, MA, October 2003).

[32] KRISTENSEN, M. D. Execution Plans for Cyber Foraging. In *MobMid '08: Proceedings of the 1st Workshop on Mobile Middleware* (Leuven, Belgium, 2008).

[33] KUMAR, P., AND SATYANARAYANAN, M. Log-based Directory Resolution in the Coda File System. In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems* (San Diego, CA, January 1993).

[34] KUMAR, P., AND SATYANARAYANAN, M. Supporting Application-Specific Resolution in an Optimistically Replicated File System. In *Proceedings of the 4th IEEE Workshop on Workstation Operating Systems* (Napa, CA, October 1993).

[35] KUMAR, P., AND SATYANARAYANAN, M. Flexible and safe resolution of file conflicts. In *Proceedings of the USENIX Winter 1995 Technical Conference* (New Orleans, LA, January 1995).

[36] LAGAR-CAVILLA, H. A., WHITNEY, J., SCANNELL, A., PATCHIN, P., RUMBLE, S., DE LARA, E., BRUDNO, M., AND SATYANARAYANAN, M. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proceedings of EuroSys 2009* (Nuremberg, Germany, March 2009).

[37] LOWE, D. The Computer Vision Industry , 2010. `http://people.cs.ubc.ca/~lowe/vision.html`.

[38] MONSOON SOLUTIONS. Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[39] MOPED. MOPED: Object Recognition and Pose Estimation for Manipulation. `http://personalrobotics.ri.cmu.edu/projects/moped.php`.

[40] MUMMERT, L. B., EBLING, M. R., AND SATYANARAYANAN, M. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating System Principles* (Copper Mountain, CO, December 1995).

[41] MYOONET. Unique Scalable Data Centers, December 2011. `http://www.myoonet.com/unique.html`.

[42] NARAYANAN, D., AND SATYANARAYANAN, M. Predictive Resource Management for Wearable Computing. In *Proceedings of the 1st International Conference on Mobile Systems Applications, and Services (MobiSys)* (San Francisco, CA, May 2003).

[43] NATH, P., KOZUCH, M., O'HALLARON, D., HARKES, J., SATYANARAYANAN, M., TOLIA, N., AND 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).

[44] OK, M., SEO, J.-W., AND PARK, M.-S. A Distributed Resource Furnishing to Offload Resource-Constrained Devices in Cyber Foraging Toward Pervasive Computing. In *Network-Based Information Systems*, T. Enokido, L. Barolli, and M. Takizawa, Eds., vol. 4658 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2007.

[45] OPENCV. OpenCV Wiki. `http://opencv.willowgarage.com/wiki/`.

[46] PAUL, R. First look: Android 4.0 SDK opens up face recognition APIs, October 2010. `http://arstechnica.com/gadgets/news/2011/10/first-look-android-40-sdk-opens-up-face\-recognition-apis.ars`.

[47] PHILLIPS, P. J., SCRUGGS, W. T., O'TOOLE, A. J., FLYNN, P., BOWYER, K. W., SCHOTT, C. L., AND SHARPE, M. FRVT 2006 and ICE 2006 Large-Scale Results. Tech. Rep. NISTIR 7408, National Institute of Standards and Technology, March 2007.

[48] PUREWAL, S. J. Siri Goes Down For a Day; Apple Says Network Outages Are Possible. *PCWorld* (November 2011). `http://www.pcworld.com/article/243175/siri_goes_down_for_a_day_apple_says_network_outages_are_possible.html`.

[49] RATTNER, E. Afghan Language Translation Devices for U.S. Army. *The Future of Things* (August 2010). http://thefutureofthings.com/news/10229/afghan-language-translation-devices-for-u-s-army.html#.

[50] RICHARDS, J. Thousands of cyber attacks each day on key utilities. *London Times* (August 2008). http://www.timesonline.co.uk/tol/news/uk/crime/article4592677.ece.

[51] SATYANARAYANAN, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications 8* (August 2001).

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

[53] SATYANARAYANAN, M., BAHL, V., CACERES, R., AND DAVIES, N. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing 8*, 4 (Oct-Dec 2009).

[54] SAVAGE, S., WETHERALL, D., KARLIN, A., AND ANDERSON, T. Practical network support for IP traceback. In *Proceedings of SIGCOMM 2000* (Stockholm, Sweden, August 2000).

[55] SAVVAS, A. Firms will flee cloud if lessons from Siri and RIM outages not learned. *CFO World* (November 2011).

[56] SCALABLE NETWORK TECHNOLOGIES. Information Warfare: Evolving Offensive and Defensive Information Warfare Strategies in Mobile Networks. *EE Times* (January 2010).

[57] SCHUDEL, G., AND WOOD, B. Adversary work factor as a metric for information assurance. In *Proceedings of the 2000 Workshop on New Security Paradigms* (Ballycotton, Ireland, 2000).

[58] SETHI, M., KANNAN, K., SACHINDRAN, N., AND GUPTA, M. Rapid Deployment of SOA Solutions via Automated Image Replication and Reconfiguration. In *Proceedings of the 2008 IEEE International Conference on Services Computing* (Washington, DC, 2008).

[59] SHAH, R. C., ROY, S., JAIN, S., AND BRUNETTE, W. Data MULEs: modeling and analysis of a three-tier architecture for sparse sensor networks. *Ad Hoc Networks 1*, 2-3 (2003).

[60] SPHINX-4. Sphinx-4: A Speech Recognizer Written Entirely in the Java Programming Language. http://cmusphinx.sourceforge.net/sphinx4/.

[61] STEERE, D. C., KISTLER, J. J., AND SATYANARAYANAN, M. Efficient user-level cache file management on the sun vnode interface. In *Summer Usenix Conference Proceedings* (Anaheim, CA, June 1990).

[62] SUD, S., WANT, R., PERING, T., ROSARIO, B., AND LYONS, K. Enabling rapid wireless system composition through layer-2 discovery. *IEEE Network 22*, 4 (July-August 2008).

[63] SULLIVAN, B. M., EVANS, B. W., AND ALLEN, P. W. Biological and Chemical Warfare Defense Sensors and Systems. *Systems and Information Technology Review Journal 8*, 1 (Spring/Summer 2000).

[64] THOMPSON, C. What is I.B.M.'s Watson? *New York Times Magazine* (June 2011). http://www.nytimes.com/2010/06/20/magazine/20Computer-t.html.

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

[66] TOLIA, N., KAMINSKY, M., ANDERSEN, D. G., AND PATIL, S. An Architecture for Internet Data Transfer. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI '06)* (San Jose, CA, May 2006).

[67] TOLIA, N., KOZUCH, M., SATYANARAYANAN, M., KARP, B., BRESSOUD, T., AND PERRIG, A. Opportunistic Use of Content-Addressable Storage for Distributed File Systems. In *Proceedings of the 2003 USENIX Annual Technical Conference* (San Antonio, TX, June 2003).

[68] WARWICK, G. DARPA Plans IED Jamming Demonstration. *Aviation Week* (August 2009). `http://www.aviationweek.com/aw/generic/story_generic.jsp?channel=defense&id=news/DARPA082709.xml&headline=DARPA%20Plans%20IED%20Jamming%20Demonstration`.

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

[70] ZIENIEWICZ, M.J., JOHNSON, D.C., WONG, D.C., FLATT, J.D. The Evolution of Army Wearable Computers. *IEEE Pervasive Computing 1*, 4 (October-December 2002).