# On-Demand VM Provisioning for Cloudlet-Based Cyber-Foraging in Resource-Constrained Environments

Sebastián Echeverría*†, James Root*, Ben Bradshaw* and Grace Lewis*
*Carnegie Mellon Software Engineering Institute
4500 Fifth Ave., Pittsburgh PA, USA
Email: {secheverria, jdroot, bwbradshaw, glewis}@sei.cmu.edu
†Universidad de los Andes
Santiago, Chile

*Abstract*—**Mobile applications are increasingly used by first responders, medics, researchers and other people in the field support of their missions and tasks. These environments have very limited connectivity and computing resources. Cloudlet-based cyber-foraging is a method of opportunistically discovering nearby resource-rich nodes that can increase the computing power of mobile devices and enhance the mobile applications running on them. In this paper we present On-Demand VM Provisioning, a mechanism for provisioning cloudlets at runtime by leveraging the advantages of enterprise provisioning tools commonly used to maintain configurations in enterprise environments. We present details of a prototype for On-Demand VM Provisioning and the results of a quantitative and qualitative evaluation of the prototype compared to other cloudlet provisioning mechanisms. The evaluation shows that On-Demand VM Provisioning shows promise in terms of flexibility, energy consumption, maintainability and leverage of cloud computing best practices, but can be challenging in disconnected environments, especially for complex applications with many dependencies.**

## I. Introduction

Mobile applications are increasingly used by first responders and other field personnel in support of their missions and tasks. These environments are not only at the edge of the network infrastructure, but are also resource-constrained due to limited computing resources, intermittent network connectivity, and in some cases dynamic context and high levels of stress. Applications that are useful to field personnel include speech and image recognition, natural language processing, sensor data collection, and situational awareness. These are all computation-intensive tasks that take a heavy toll on the device's battery power and computing resources.

Cyber-foraging is the leverage of external resource-rich surrogates to augment the capabilities of resource-limited mobile devices [1]. Most existing cyber-foraging solutions rely on conventional Internet for connectivity to the cloud or strategies that tightly couple mobile clients with servers at deployment time. These solutions are typically not appropriate for resource-constrained environments because of their dependence on multi-hop networks to the cloud and static deployments. Cloudlet-based cyber-foraging relies on discoverable, generic, stateless servers located in single-hop proximity of mobile devices. Applications that leverage cloudlet-based cyber-foraging are typically set up as a thin client that runs on the mobile device and a computation-intensive server that runs on the cloudlet. At runtime, once an appropriate cloudlet has been discovered by a mobile device it has to be set up to provide the capabilities that are needed; we call this *cloudlet provisioning*.

Provisioning tools are commonly used in enterprises to set up and maintain the configurations of the computers owned by an organization [2]. These tools are used to quickly set up a computer/server with all the components that are required for the machine to provide a predefined computing environment (i.e., organizational computing baselines). They are usually also used to ensure that the required state and configuration is maintained over time as changes are made to configurations and components. Some of these tools also work at a lower level, helping with the task of setting up virtual machines according to a certain configuration [3]. This type of tool could be used in cyber-foraging solutions, in particular for cloudlet provisioning. However, little research has been done in the use of provisioning tools at smaller scale and potentially in resource-constrained and disconnected environments.

The goal of this paper is to present and analyze a mechanism called *On-Demand VM Provisioning* that leverages the best practices that provisioning tools offer to enterprise environments by adapting them to the cyber-foraging context. These advantages include automation of the setup process, flexibility in the selection and inclusion of components in a virtual machine (VM), and the use of mature tools with proven capabilities and support.

The remainder of this paper is organized as follows. Section II presents a summary of related work in cyber-foraging and provisioning tools. Section III describes cloudlet-based cyber-foraging and the requirements that cloudlet-based solutions have to take into account. Section IV describes the On-Demand VM Provisioning mechanism and the challenges of using enterprise provisioning tools in resource-constrained environments. Section V describes the evaluation and selection process that we followed to select an appropriate provisioning tool. Section VI presents the architecture and design of the On-Demand VM Provisioning prototype for cloudlet provisioning. Section VII presents the results of the system evaluation. Finally, Section VIII presents conclusions and future research directions.

978-1-63190-024-2 © 2014 ICST
DOI 10.4108/icst.mobicase.2014.257768

## II. Related Work

Previous work on cyber-foraging has presented different approaches on how to partition code to be offloaded to a remote server. Systems like MAUI [4] offload only specific methods, while others like CloneCloud [5] work at the process level by offloading threads. Other approaches work at higher levels of abstraction by offloading complete applications along with their environments [1][6][7]. These systems also differ on when they define what to offload. Some systems decide what to offload at runtime [4][5], while others compose applications in such a way that the offloadable pieces are defined at design and implementation time [1][7].

The work that is most similar to that proposed in this paper is cyber-foraging systems in which setup instructions are provided to the offload target. In the Collective Surrogates system [8] the mobile device sends a small program which is simply a script that offloads code from the Internet, installs, and runs it. In the MAPCloud system [9] an application request is modeled as a workflow of tasks. The offload target (which in this case acts as a broker) locates other offload targets that can perform the tasks and returns a service plan with the URL of each offloaded workflow task. To the best of our knowledge, work on cyber-foraging has not focused on on-the-fly assembly of offloaded computation using provisioning tools.

Provisioning tools are commonly used in enterprises to manage the configuration of real or virtual machines [10]. Managing the configuration of different components of a system used to require different tools for different types of components, but more recent tools are able to configure a complete environment on top of a specific operating system [11]. Configuration management tools, for example, allow system administrators to easily create copies of an existing machine or of a pre-defined environment described in a configuration script [11][12]. Research on system administration and configuration management has focused on issues such as formal analysis of system administration [13], simplifying the configuration process and tools [14], and deploying system configurations onto virtual machines [3]. Tools such as Vagrant [15] simplify the creation and management of VMs by integrating configuration management tools that can automatically provision a newly created VM. Research on opportunistically provisioning VMs focuses on reducing costs or access times by optimizing the use of resources when multiple VMs can execute concurrently [16][17][18]. However, most of this work assumes that the environment for these provisioning tools consists of enterprise networks with good connectivity and access to resources.

## III. Cloudlet-Based Cyber-foraging

Cloudlets are discoverable, generic, stateless servers located in single-hop proximity of mobile devices, that can operate in disconnected mode and are virtual-machine (VM) based to promote flexibility, mobility, scalability, and elasticity [1]. In the cloudlet architecture we have developed, applications are partitioned at design-time into two parts: a thin client called a *Cloudlet-Ready App* that executes on a mobile device, and a computation-intensive *Application Server* that runs inside a VM on the cloudlet server [19]. The Application Server provides a service to the Cloudlet-Ready App. The VM that hosts this service is called a *Service VM* [20].

Cloudlet provisioning is the process of setting up a Service VM on a cloudlet so that a mobile device can have access to the service it provides. An effective provisioning technique has to be able to work efficiently in the resource-constrained environments in which cloudlets may operate. We define the following quality attributes [21] to measure the usefulness of a cloudlet provisioning technique for resource-constrained environments:

- **Energy efficiency**: Refers to how much energy a mobile device consumes when provisioning a cloudlet. Due to the limited capacity of the batteries used by mobile devices, reducing the amount of energy consumed during provisioning is a priority. Given that wireless transmission accounts for a large part of the battery drain on this type of devices [22], battery life on the mobile device can be extended by decreasing the amount of information sent through the network radio for provisioning a cloudlet.
- **Application-ready time**: We define this as the time between start of cloudlet provisioning and acknowledgment of the Application Server that it has started. Smaller application-ready times provide a better user experience because it takes less time to use a mobile Cloudlet-Ready App. In addition, due to the dynamic context and urgency of some resource-constrained environments users may not have the luxury of waiting for long periods of time before being able to use a mobile application. Provisioning techniques need to account for this and should strive to reduce application-ready time.
- **Automation of provisioning**: Refers to how much manual work is required to provision new Service VMs on a cloudlet. Cloudlets at the edge are not in dedicated data centers, can be mobile, and are in an environment where there may not be skilled system administrators, or even time for administrators to monitor them. Cloudlets have to be able to function with as little human administration as possible and therefore provisioning techniques have to be as self-sufficient as possible.
- **Flexibility**: Refers to how adaptable the provisioning mechanism is to changes in the configuration of the cloudlet. At the edge, cloudlets may need to be quickly replaced, and mobile devices will need to connect to different cloudlets depending on their availability. Provisioning techniques should be able to work correctly with different cloudlets.

## IV. On-Demand VM Provisioning

On-Demand VM Provisioning uses a provisioning script at runtime to set up a Service VM. The mobile device sends the provisioning script to the cloudlet when the user executes a Cloudlet-Ready App that needs access to the service. The cloudlet executes the script inside a clean VM (a *Baseline VM*) and uses a provisioning tool to create a Service VM that has all the components that it needs to provide the service.

The environment for which provisioning tools are designed has some substantial differences with the resource-constrained environments in which cloudlets may operate. Part of the goal of our prototype is to be able to find ways to overcome these differences. In particular:

- Provisioning tools deployed on an enterprise network rely on an infrastructure of servers to achieve their goals.

Central servers are commonly used to maintain common configurations that are deployed on nodes. There is usually connectivity to remote Internet servers or to internal repositories of components. Cloudlets, on the other hand, are expected to work on very small networks, usually composed of the cloudlet and a mobile device only, with no permanent infrastructure (cloudlets can move around and connect to different devices). They have very limited connectivity to the Internet or to other servers.

- On enterprise networks there is a substantial amount of manual work performed by data center administrators whose job is to ensure that the network is working properly. Cloudlets are expected to work with very limited human supervision, especially in resource-constrained environments in which intermittent connectivity makes it difficult for continuous monitoring of their state. Cloudlets require a more robust automated working mode because administrators will only seldom be available to monitor the status of a cloudlet and to manually fix issues.
- Deployment of new capabilities can be a carefully planned and orchestrated task on enterprise networks. Cloudlets, on the other hand, have to be able to quickly react to requirements from mobile devices and assemble services on the cloudlet on-the-fly. Provisioning tools on cloudlet environments have to be able to easily and automatically work with different Application Servers that need to be set up on a cloudlet.

To design our On-Demand VM Provisioning prototype, the first step was to find a suitable provisioning tool that would offer enough flexibility to address the challenges mentioned above. The next step was to design the prototype to use this tool in such a way that it could handle the characteristics of cloudlet environments properly.

## V. Provisioning Tools

A major design decision for On-Demand VM Provisioning was to select an appropriate provisioning tool to set up a Service VM. We defined a set of requirements that an existing provisioning tool should address:

- Support for Windows and Linux operating systems because these are the operating systems used by our benchmark applications and would also enable reuse of many existing applications.
- Disconnected operation mode so that it can work on a cloudlet that is on an isolated network (i.e., does not depend on a remote service and does not need to execute on a cloud platform).
- Stand-alone execution because Service VMs are transient and are created and disposed of at runtime (i.e., no need for a central server to maintain configurations for long periods of time).
- On-demand execution so that it can be executed immediately when the VM is provisioned (as opposed to executing at some fixed interval).
- Declarative mechanism for defining the components that should be part of a VM (i.e., defines the end state of the VM rather than the exact steps to follow to configure the VM), to simplify the creation of provisioning scripts.
- Support for starting any type of executable after provisioning the Service VM so that the existing Application Servers used in our benchmarks can be used without modifications (and in general, to make it easy for existing servers to be used without having to make major changes).

The tools that we surveyed fall in one of these categories:

- System Configuration Management Software (SCM): Tools that can install an operating system, install dependencies, and configure them so that services run adequately. These tools store the configurations so that they can be reused and updated over time as required. They usually also allow administrators to manage the state of multiple machines remotely. Examples include Puppet[1], Chef[2], CFEngine[3], Bcfg2[4], SmartFrog[5] and Salt[6]. Most of them have a client-server design, with multiple clients installed on nodes that pull the configuration for their host from the server to update the environment to match the configuration. Other tools that take different approaches include Docker[7] (a Linux container engine that creates process-level containers to provide an isolated environment) and NixOS[8] (a Linux distribution that uses a purely-functional package manager to define configurations).
- Service Orchestration Software (SO): Tools that simplify the configuration of services, their relations, and the way they should scale.An example is JuJu[9], which is targeted at provisioning services in the cloud and easy integration and scaling of services.
- Virtual Machine Management Software (VMM): Tools that simplify the creation, configuration, execution and maintenance of VMs. An example is Vagrant[10], a tool that creates a VM with a base VM image (box), executes predefined provisioning scripts, and launches the VM. It can integrate with SCM tools such as Puppet and Chef.

We quickly found that most tools focus on provisioning existing machines instead of setting up new VMs and then provisioning them. Even though several tools matched our criteria, Puppet and Chef appeared to be the most mature. We selected Puppet [23] because of its larger community and its slightly simpler declaration language. The best choice would have been to use Vagrant (to set up transient VMs) in combination with Puppet. However, because Vagrant did not support the QEMU virtual machine manager, on which our prototype is based, we had to discard it.

## VI. System Architecture

Figure 1 shows the high-level architecture of the system. The architecture of this prototype is designed to support different provisioning mechanisms to set up a Service VM on a cloudlet. The mobile device carries the files that it needs to provision the cloudlet. However, the format and

[1]http://puppetlabs.com/puppet/puppet-open-source
[2]http://www.getchef.com/chef/
[3]http://cfengine.com/
[4]http://bcfg2.org/
[5]http://www.smartfrog.org
[6]http://www.saltstack.com/
[7]https://www.docker.com/
[8]http://nixos.org/
[9]https://juju.ubuntu.com/
[10]http://www.vagrantup.com/

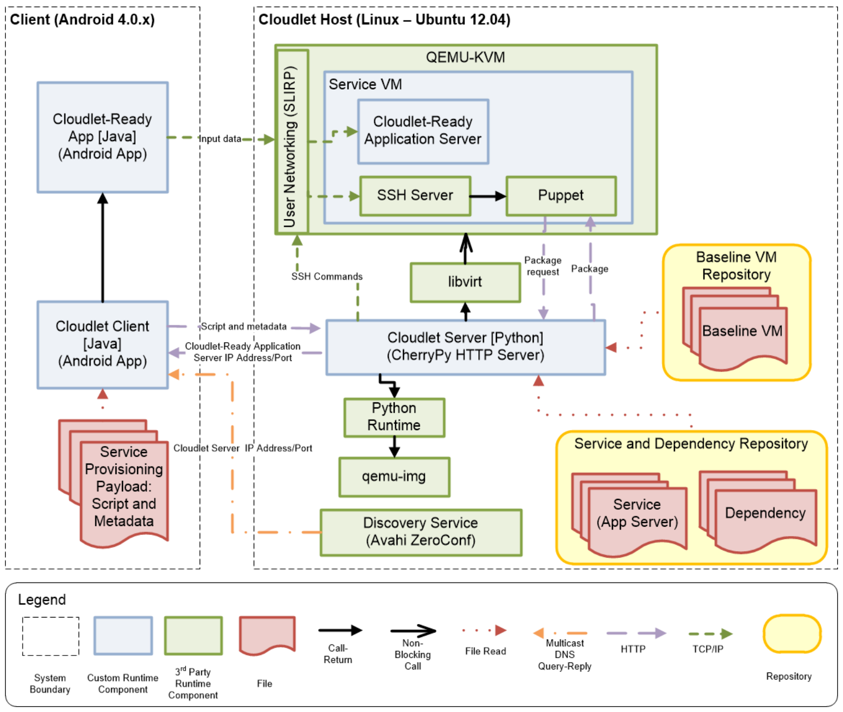

Client (Android 4.0.x)
Cloudlet Host (Linux – Ubuntu 12.04)
QEMU-KVM
Service VM
User Networking (SLIRP)
Cloudlet-Ready App [Java] (Android App)
Input data
Cloudlet-Ready Application Server
SSH Server
Puppet
libvirt
SSH Commands
Package request
Package
Baseline VM Repository
Baseline VM
Cloudlet Client [Java] (Android App)
Script and metadata
Cloudlet-Ready Application Server IP Address/Port
Cloudlet Server [Python] (CherryPy HTTP Server)
Python Runtime
qemu-img
Discovery Service (Avahi ZeroConf)
Cloudlet Server IP Address/Port
Service Provisioning Payload: Script and Metadata
Service and Dependency Repository
Service (App Server)
Dependency
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Call-Return
Non-Blocking Call
File Read
Multicast DNS Query-Reply
HTTP
TCP/IP
Repository

Fig. 1. High-level architecture of the cloudlet-based cyber-foraging prototype that implements On-Demand VM Provisioning.

content of these files depend on the cloudlet provisioning mechanism being used. The provisioning process is controlled by a generic *Cloudlet Client* application on the mobile device that communicates with a *Cloudlet Server* application running on the cloudlet. Upon receiving the provisioning files, the Cloudlet Server sets up the Service VM and makes its service available to the *Cloudlet-Ready App*, which can then interact directly with the *Cloudlet-Ready Application Server* (*Application Server* for short). Details of the common components of the architecture that are shared between cloudlet provisioning mechanisms can be found in [20].

## A. Main Components

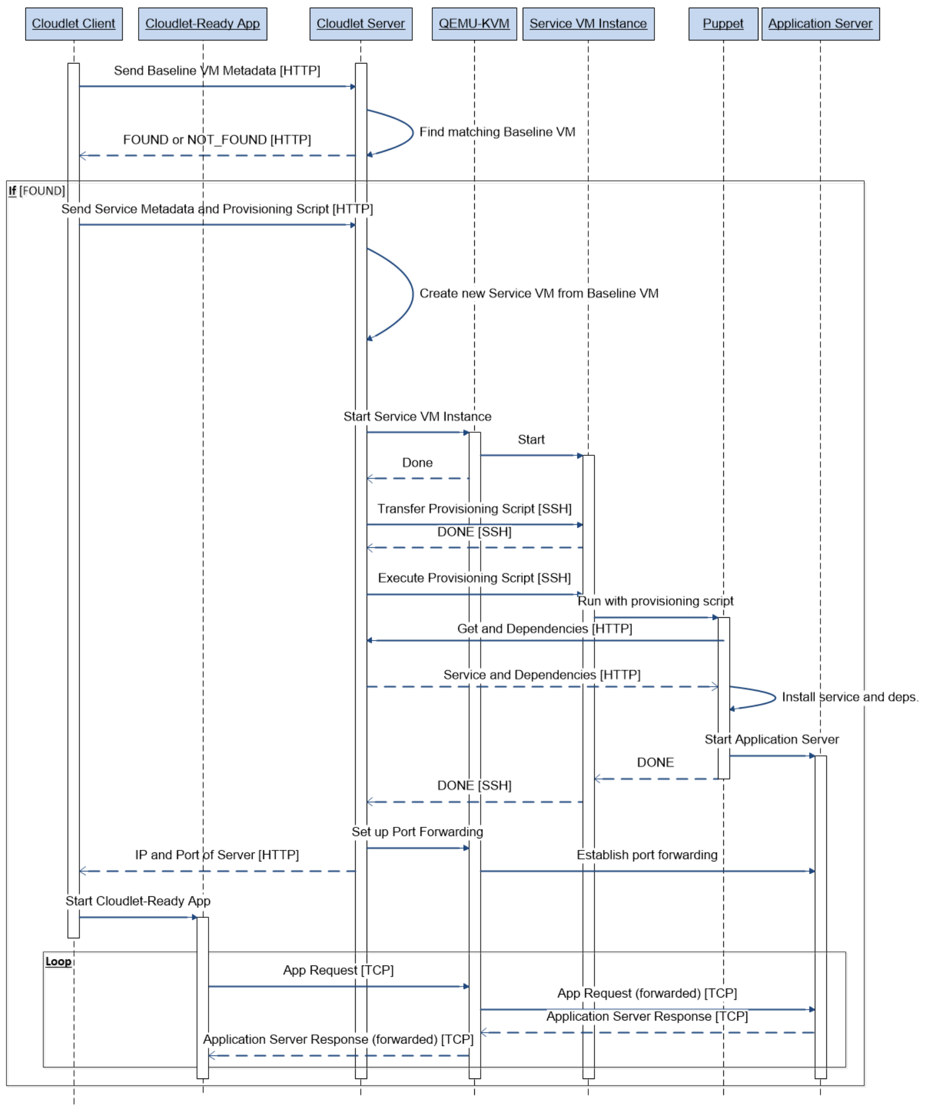
In Figure 1, the components that are specific to On-Demand VM Provisioning are the Baseline VM Repository, the Service and Dependency Repository, the Service Provisioning Payload, the Service VM SSH Server, and Puppet. What follows is a description of these components and how they work together.

*1) Baseline VM Repository:* A *Baseline VM* is a suspended VM that has all the components that are considered part of a baseline configuration. A Baseline VM can be used as a template for the provisioning of Service VMs. Baseline VMs can be modified, updated, and maintained continuously without affecting the provisioning process. Changes to a Baseline VM will only affect new Service VMs that are derived from it because there is no persistent link between a Service VM and the Baseline VM that it was created from.

In our prototype, Baseline VMs are created with the virtual machine manager QEMU [24], plus the KVM kernel module. Service VMs created from Baseline VMs are set up with User Networking [25], which isolates the VM in an internal virtual network contained inside the QEMU process that is hosting the VM. Services that need to be accessed from outside the VM, such as the ones provided by the Application Server, are mapped through QEMU's port forwarding configuration.

Baseline VMs are stored in the Baseline VM Repository as a set of files with the following structure:

- Disk Image file (.qcow2): Disk image file in QCOW2 format [26] used as the virtual disk of a VM. It contains a basic OS installation plus common libraries that will likely be used by many services. A Baseline VM will need to have at least the following components installed or configured in its disk image, according to our current implementation:
  - SSH server: Enables the Cloudlet Server to send files and commands. The SSH port is forwarded though QEMU so that the Cloudlet Server can connect through SSH.
  - Puppet client: Enables the execution of Puppet manifests inside the VM as a standalone component (no master/agent setup required).
  - Link to Cloudlet HTTP File Server: Enables the Service VM to download packages stored locally on the Cloudlet (through HTTP). With User Networking, the Service VM always uses the same virtual IP address to contact the VM host. The Baseline VM has to be configured once to be able to reach this IP address when installing packages.
- VM State Image file (.lqs): VM state image generated by the libvirt VM management API [27]. This image includes both the description of the VM that was suspended, as well as the memory state of the VM.
- Baseline VM Metadata file (.jsonbmd): JSON file describing the basic features of the Baseline VM. It has the following fields:



Cloudlet Client
Cloudlet-Ready App
Cloudlet Server
QEMU-KVM
Service VM Instance
Puppet
Application Server
Send Baseline VM Metadata [HTTP]
Find matching Baseline VM
FOUND or NOT_FOUND [HTTP]
If [FOUND]
Send Service Metadata and Provisioning Script [HTTP]
Create new Service VM from Baseline VM
Start Service VM Instance
Start
Done
Transfer Provisioning Script [SSH]
DONE [SSH]
Execute Provisioning Script [SSH]
Run with provisioning script
Get and Dependencies [HTTP]
Service and Dependencies [HTTP]
Install service and deps.
Start Application Server
DONE
DONE [SSH]
Set up Port Forwarding
IP and Port of Server [HTTP]
Establish port forwarding
Start Cloudlet-Ready App
Loop
App Request [TCP]
App Request (forwarded) [TCP]
Application Server Response [TCP]
Application Server Response (forwarded) [TCP]

Fig. 2. Sequence diagram for cloudlet provisioning and Cloudlet-Ready App execution using On-Demand VM Provisioning.

- ◦ osFamily (string): OS family (e.g., ”Linux”, ”Windows”).
- ◦ os (string): Name of the specific OS or distribution (e.g., ”Windows 7”, ”Ubuntu”).
- ◦ osVersion (string): Version of the OS (e.g., ”SP1”, ”8.1”, ”12.10”).
- ◦ osISA (string): Instruction set of the compiled OS in the disk image (e.g., ”x86-32”, ”x86-64”).

A Baseline VM Repository contains a folder for each Baseline VM that it stores. The name of the folder is a unique ID that is used to identify the Baseline VM. Each folder has the three files described above.

*2) Service Provisioning Payload:* The Service Provisioning Payload files describe how to provision a Service VM on a cloudlet. They are stored on the mobile device and are transferred to the cloudlet when the client needs access to the service provided by that Service VM. The payload is composed of two metadata files and a provisioning script:

- Baseline VM Metadata file (.jsonbmd): JSON file that describes the features that are required of a Baseline VM to serve as a template to create an appropriate Service VM. The format is the same as the Baseline VM Metadata file described above, although some fields can be omitted if there is no requirement related to them.
- Service Metadata file (.jsonsvm): JSON file that describes the attributes of a service to be hosted on the Service VM that will be provisioned. These attributes include:
  - ◦ serviceId (string): Unique identifier of the service that will be provided by the Service VM.
  - ◦ servicePort (integer): Port that the Application Server will be listening on to provide its service inside the Service VM.
- Puppet Manifest (text file): Script detailing what has to be provisioned to set up the Service VM. This includes the Application Server code to install, dependencies for that server, and required libraries. It is written in the default language used for Puppet manifests (http://docs.puppetlabs.com/learning/manifests.html).

*3) Service and Dependency Repository:* A cloudlet host has a repository of packages that are available to a Service VM to be able to provision itself. There are two types of packages:

- Service Packages: Application Server files that provide the actual services, packaged in an easy-to-install way that can be handled by Puppet.
- Dependency Packages: Components or libraries that can be used by different services.

In the prototype, all these packages are made available via HTTP from the Cloudlet Server because it can serve static files via HTTP. The Ubuntu packages are available in an Apt-Get [28] repository installed inside the Cloudlet Host. The Windows components are stored as Windows Installer Packages (MSI) [29] inside the same Cloudlet Host.

### B. Provisioning Sequence

The provisioning process is shown in Figure 2 and follows these steps:

1) The Cloudlet Client sends the Baseline VM Metadata file (part of the Service Provisioning Payload) to the Cloudlet Server to look for a matching Baseline VM.
2) The Cloudlet Server checks if the cloudlet has a Baseline VM that matches the requirements given in the Baseline VM Metadata file.
3) If it does, the Cloudlet Client sends the Service Metadata file and the Provisioning Script (part of the Service Provisioning Payload) to the Cloudlet Server.
4) The Cloudlet Server creates a copy of the Baseline VM, sets it up using the Service Metadata and starts it as a new Service VM through QEMU.
5) The Cloudlet Server transfers the Provisioning Script (Puppet Manifest) via SSH to the running Service VM instance and sends an SSH command for Puppet to execute the script.
6) Puppet, inside the Service VM Instance, executes the Provisioning Script.
7) Puppet obtains the files for the service and its dependencies from the Cloudlet Host using HTTP download via the Cloudlet Server.
8) Puppet installs the service and its dependencies, starts the Application Server that provides the service, and notifies the Cloudlet Server that the installation is complete.
9) The Cloudlet Server sends the IP address and port that will be used to connect to the Service VM back to the Cloudlet Client.
10) The Cloudlet Client starts up the Cloudlet-Ready App that will access the service.
11) The Cloudlet-Ready App communicates with the Application Server through the forwarded port set up by the QEMU process hosting the Service VM.

## VII. System Evaluation

This section focuses on the evaluation of the On-Demand VM Provisioning System against the requirements of cyber-foraging in resource-constrained environments. It starts by describing previous cloudlet provisioning mechanisms that we have developed as part of our research. It then presents the analysis of the architecture of the prototype as well as some experimental results.

### A. Previous Cloudlet Provisioning Mechanisms

We implemented several cloudlet provisioning mechanisms as part of our research [20]. Part of the motivation to design and implement the On-Demand VM Provisioning approach was to try to overcome some of the shortcomings of these methods.

*VM Synthesis* works by creating an overlay, which is the binary difference between a Base VM and a Service VM (which is created by installing an Application Server on the Base VM and calculating the binary difference between the two image files). The overlay is carried by the mobile device and transferred to a cloudlet to reassemble the full Service VM if it has the same Base VM. The main advantage of VM Synthesis is that it ensures the proper execution of the Application Server because it packages the full environment that the server needs to run on. However, the overlay tends to be significant in size, and transferring it to the cloudlet consumes a large amount of energy from the mobile device. Also, VM Synthesis is not very flexible in its deployment and maintenance because the exact Base VM has to exist in the cloudlet to reassemble the Service VM (no security updates can be added to the base VM, for example).

*Cached VM* works by pre-provisioning a cloudlet with full Service VM images. Each VM image file has a unique service identifier. At runtime, the mobile device instructs the cloudlet to start the VM that corresponds to the service for the launched client app. This method requires almost no communication between the mobile device and the cloudlet, other than sending the identifier of the required Service VM. However, Service VMs have to be provisioned in advance on any cloudlet that the mobile device may connect to.

In *Cloudlet Push*, the cloudlet is not only pre-provisioned with Service VM images, but also the corresponding mobile client apps. At runtime, the mobile device queries the cloudlet for available capabilities, similar to accessing an app store. The cloudlet pushes the selected client app to the mobile device and then starts the corresponding Service VM. This has the same advantages and disadvantages as the previous method, plus the issue of ensuring that the apps stored in the cloudlet are compatible with the device that is requesting them.

### B. Qualitative Evaluation: Analysis of the Architecture

The following is an analysis of how On-Demand VM Provisioning addresses the cloudlet requirements we defined in Section III, in comparison to the other cloudlet provisioning mechanisms described above.

- **Energy efficiency**: On-Demand VM Provisioning needs to transfer only a small script and some metadata to provision a Service VM on a cloudlet. VM Synthesis has to send much more data during the provisioning process, including (in binary difference format) the Application Server itself and its dependencies, and the memory state of the VM. On the other hand, Cached VM and Cloudlet Push do not require to send any data at all (other than an identifier) during provisioning, mainly because they assume that the cloudlet has already been provisioned. Of the techniques that actually provision a cloudlet, On-Demand VM Provisioning transfers the least amount of data.

TABLE I. PAYLOAD SIZE, APPLICATION-READY TIMES AND CONSUMED CLIENT ENERGY FOR ON-DEMAND VM PROVISIONING (AVERAGES).

| Application | Payload Size (KB) | Application-Ready Time (s) | Total Client Energy (J) | Client Comm. Energy (J) |
|---|---|---|---|---|
| **FACE (Windows)** | 0.68 | 112.7 | 129.1 | 16.4 |
| **OBJECT (Linux)** | 1.23 | 211.0 | 244.0 | 33 |
| **SPEECH (Win.)** | 1.32 | 237.6 | 269.2 | 31.6 |
| **SPEECH (Linux)** | 0.76 | 94.1 | 109.3 | 15.2 |

- **Application-ready time**: Because the VM has to be provisioned before it can provide a service, the time that it takes to be ready will vary depending on the time that Puppet takes to configure the system. For a service with multiple dependencies, each of these would have to be transferred and installed in the VM before it is ready to be used by the mobile client. In comparison, the application-ready time for VM Synthesis depends on the time that it takes to transfer the payload and set up the Service VM, which could be less than the time that On-Demand VM Provisioning may take for complex services with many dependencies. Cached VM and Cloudlet Push have almost no application-ready time, since they assume that the Service VM is already provisioned. Cached VM could be used to cache VMs assembled by On-Demand VM Provisioning to decrease the application-ready time for new requests for the same service.
- **Automation of provisioning**: Provisioning a Service VM through On-Demand VM Provisioning is done with no user intervention on the cloudlet side. However, it does need the cloudlet to be have Baseline VMs and a repository of services and dependencies already set up. VM Synthesis, in comparison, needs only the correct Base VMs to be set up before the provisioning process and the provisioning process itself is also automated. On the other hand, Cached VM and Cloudlet Push need manual provisioning of some sort to work on their own because they assume that the cloudlet administrator will have already provisioned the necessary Service VMs.
- **Flexibility**: Because a provisioning script only defines the basic features of a Baseline VM and the dependencies that are needed, these components can be maintained and updated without any negative effects on the On-Demand VM Provisioning process. Baseline VMs can be safely upgraded and patched, and new versions of libraries can be made available on the cloudlet. VM Synthesis, in comparison, needs the exact Base VM used to create an overlay, which cannot be modified without having to re-create all overlays generated from it. There is a trade-off, however, between the flexibility of On-Demand VM Provisioning and the probability of failure due to a missing dependency. With On-Demand VM Provisioning, if a dependency or library that is required by the service is not available on the cloudlet, it will not be possible to set up a Service VM. By allowing more flexibility, there are more ways for the process to fail in comparison to other cloudlet provisioning mechanisms (VM Synthesis, for example, only needs the Base VM). A potential way of overcoming this issue is to allow the cloudlet to obtain dependencies from the cloud when Internet connectivity is available. The cloudlet could asynchronously download more dependencies to increase the chances of having all the dependencies needed for a particular service.

### *C. Quantitative Evaluation: Experiments and Results*

We conducted a set of experiments to compare On-Demand VM Provisioning against our previous provisioning techniques. For the experiments we used three applications that were partitioned into Cloudlet-Ready Apps and Application Servers: face recognition (FACE), speech recognition (SPEECH), and object recognition (OBJECT). These are three Android-based apps we developed internally for testing purposes. We used a Galaxy Nexus with Android 4.3 as a mobile device and a Core i7-3960x based server with 32 GB of RAM running Ubuntu 12.04 as the cloudlet. We created a self-contained wireless network (using Wi-Fi 802.11n at 2.4 GHz, 65 Mbps) to be able to isolate network traffic effects. Energy was measured using a PowerMeter from Monsoon Solutions. For details on the complete experiments, see [20].

In the data tables, application-ready time is measured as the time in seconds from the start of the provisioning process until the cloudlet responds that it is ready. Total client energy is measured as the total energy consumed on the mobile device during application-ready time. Client communication energy is calculated by subtracting the energy consumed by the phone while idle (measured as 1 J/s in our experiments) from the total client energy consumed (to approximate the actual energy spent by On-Demand VM Provisioning, discarding other energy consumers such as the screen). This last value was indirectly calculated and not measured because the power monitor measures total energy consumption, and does not distinguish between energy consumed for communication from energy consumed by other parts of the mobile device.

Application-ready time is very variable for On-Demand VM Provisioning, as can be seen in Table I. The Windows version of SPEECH has a much longer application-ready time than its Linux counterpart because in Windows the component installation processes have more steps. The OBJECT test app is the one with the highest number of dependencies, increasing its application-ready time. The client energy consumed is directly proportional to the application-ready time. The approximate energy consumed by the transfer of messages and the payload is proportional to the payload size (the scripts and and metadata transferred).

TABLE II. APPLICATION-READY TIMES OF OTHER CLOUDLET PROVISIONING MECHANISMS AS A PERCENTAGE OF THE TIMES FOR ON-DEMAND VM PROVISIONING (AVERAGES)

| | Application-Ready Time | | |
|---|---|---|---|
| Application | VM Synthesis | Cached VM | Cloudlet Push |
| **FACE (Windows)** | 47% | 7% | 7% |
| **OBJECT (Linux)** | 83% | 5% | 6% |
| **SPEECH (Windows)** | 36% | 5% | 5% |
| **SPEECH (Linux)** | 105% | 13% | 14% |

TABLE III. CLIENT ENERGY CONSUMED BY COMMUNICATIONS BY OTHER CLOUDLET PROVISIONING MECHANISMS AS A PERCENTAGE OF THE ENERGY CONSUMED BY COMMUNICATIONS BY ON-DEMAND VM PROVISIONING (AVERAGES)

| | Client Energy | | |
|---|---|---|---|
| **Application** | **VM Synthesis** | **Cached VM** | **Cloudlet Push** |
| **FACE (Windows)** | 27% | 13% | 36% |
| **OBJECT (Linux)** | 478% | 6% | 16% |
| **SPEECH (Windows)** | 284% | 8% | 17% |
| **SPEECH (Linux)** | 484% | 18% | 36% |

Table II shows that the application-ready times are lower for Cached VM and Cloudlet Push in comparison to On-Demand VM Provisioning. This is expected because these two methods have the Service VM already provisioned on the cloudlet. VM Synthesis shows lower application-ready times in comparison to On-Demand VM Provisioning, for the most part. This most likely has to do with the amount of dependencies required by the benchmark applications, which in turn make On-Demand VM Provisioning take longer to assemble the Service VM. On-Demand VM Provisioning is a bit faster than VM Synthesis in the case of the SPEECH test application in Linux because this is the benchmark server with the least dependencies.

Table III shows that, as expected, the client energy consumed is lower for Cached VM because it has to transfer no payload. On-Demand VM Provisioning consumes more energy than Cloudlet Push, which has to transfer a packaged app from the cloudlet server to the mobile device. Even though the app is bigger than the payload of On-Demand VM Provisioning, the energy consumed on the mobile device is lower because it is only receiving data and not sending data [30]. On-Demand VM Provisioning consumes much less energy for communications than VM Synthesis in most cases, due to the large amount of data being transferred by this cloudlet provisioning mechanism.

## VIII. Conclusions and Future Work

On-Demand VM Provisioning is a valid and effective alternative to previous cloudlet provisioning mechanisms. It leverages the flexibility and automation capabilities of a configuration management tool such as Puppet to address the issues of energy efficiency, application-ready time, automation and robustness when working in resource-constrained edge environments.

This technique does have some drawbacks, mainly related to the application-ready time and robustness for complex applications with multiple dependencies. If the scope and type of applications using this system is clearly defined, however, these problems may not be a major issue.

A hybrid mechanism could be used to leverage the advantages of the different cloudlet provisioning techniques. A cloudlet could be set up to have all these cloudlet provisioning mechanisms available, and the system could select the one that is the most appropriate based on the current context (e.g., remaining battery, available bandwidth, cloudlet connectivity). For example, if the mobile device has very little battery power remaining, and the cloudlet is temporarily connected to the Internet, On-Demand VM Provisioning could be selected to provision the cloudlet because it would require very little energy consumption from the device, and any missing dependencies could be downloaded from the Internet.

Another similar approach would be to combine the different provisioning mechanisms by attempting to use each of them sequentially, falling back on the next one if the previous one fails to provision the cloudlet. For example, Cached VM could be tried first, but it would only work if the cloudlet already has the Service VM in its cache. If this is not the case, On-Demand VM Provisioning could be used next, and if it does not work out due to missing dependencies, VM Synthesis could be attempted as a last resort.

In any case, keeping a cache of assembled Service VMs is a good idea to avoid sending data for a service that is already available. Once a Service VM is provisioned by On-Demand VM Provisioning, it can be stored on the cloudlet's internal Service VM cache. The next time a mobile device requests the service, the Cloudlet Server can first check if the Service VM is already cached, and only perform On-Demand VM Provisioning if it is not. It may be necessary to re-assemble a Service VM for a particular service if a new version of the service is available, or new dependencies are used by it. Information such as service version could be added to the service metadata file to force the re-assembly of an outdated Service VM through On-Demand VM Provisioning.

Future work could also focus on defining which provisioning mechanism is more appropriate for an application based on its characteristics. Applications that have very few dependencies can benefit considerably from On-Demand VM Provisioning, and therefore this type of application could be packaged to use this technique. Complex or dependency-heavy applications may be better handled by other techniques, such as VM Synthesis. A cloudlet could then choose which provisioning mechanism to use based on the characteristics of the application, or on a preference explicitly provided by the application.

## Acknowledgements

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution (DM-0001547).

## References

[1] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, Oct 2009.

[2] J. Rahman, "Investigating configuration management tools usage in large infrastructure," Master's thesis, University of Oslo, Department of Informatics, 2012.

[3] G. Vallee, T. Naughton, and S. L. Scott, "System management software for virtual environments," in *Proceedings of the 4th International Conference on Computing Frontiers*, ser. CF '07. New York, NY, USA: ACM, 2007, pp. 153–160. [Online]. Available: http://doi.acm.org/10.1145/1242531.1242555

[4] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62. [Online]. Available: http://doi.acm.org/10.1145/1814433.1814441

[5] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 301–314. [Online]. Available: http://doi.acm.org/10.1145/1966445.1966473

[6] A. Iyer and T. Roopa, "Extending android application programming framework for seamless cloud integration," in *Mobile Services (MS), 2012 IEEE First International Conference on*, June 2012, pp. 96–104.

[7] C. Jarabek, D. Barrera, and J. Aycock, "Thinav: Truly lightweight mobile cloud-based anti-malware," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 209–218. [Online]. Available: http://doi.acm.org/10.1145/2420950.2420983

[8] S. Goyal, "A collective approach to harness idle resources of end nodes," Ph.D. dissertation, University of Utah, 2011.

[9] M. R. Rahimi, N. Venkatasubramanian, S. Mehrotra, and A. V. Vasilakos, "Mapcloud: Mobile applications on an elastic and scalable 2-tier cloud architecture," in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 83–90. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.25

[10] S. Pandey, "Investigating community, reliability and usability of cfengine, chef and puppet," Master's thesis, University of Oslo, Department of Informatics, 2012.

[11] E. Dolstra, M. Bravenboer, and E. Visser, "Service configuration management," in *Proceedings of the 12th International Workshop on Software Configuration Management*, ser. SCM '05. New York, NY, USA: ACM, 2005, pp. 83–98. [Online]. Available: http://doi.acm.org/10.1145/1109128.1109135

[12] C. Lueninghoener, "Getting started with configuration management," *;login:*, vol. 36, no. 2, pp. 12–17, Apr 2011.

[13] M. Burgess, "On the theory of system administration," *Science of Computer Programming*, vol. 49, no. 1, pp. 1–46, 2003.

[14] A. Sekiguchi, K. Shimada, Y. Wada, A. Ooba, R. Yoshimi, and A. Matsumoto, "Configuration management technology using tree structures of ict systems," in *Proceedings of the 15th Communications and Networking Simulation Symposium*, ser. CNS '12. San Diego, CA, USA: Society for Computer Simulation International, 2012, pp. 4:1–4:7. [Online]. Available: http://dl.acm.org/citation.cfm?id=2331762.2331766

[15] J. Palat, "Introducing vagrant," *Linux J.*, vol. 2012, no. 220, Aug. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2371484.2371486

[16] M. Bjorkqvist, L. Chen, and W. Binder, "Opportunistic service provisioning in the cloud," in *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 237–244.

[17] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*. Washington, DC, USA: IEEE Computer Society, Oct 2009, pp. 50–57.

[18] K. Wang, J. Rao, and C.-Z. Xu, "Rethink the virtual machine template," in *Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, ser. VEE '11. New York, NY, USA: ACM, 2011, pp. 39–50. [Online]. Available: http://doi.acm.org/10.1145/1952682.1952690

[19] S. Simanta, G. A. Lewis, E. Morris, K. Ha, and M. Satyanarayanan, "A reference architecture for mobile code offload in hostile environments," in *Proceedings of the Joint Working IEEE/IFIP Conference Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 282–286.

[20] G. A. Lewis, S. Echeverría, S. Simanta, B. Bradshaw, and J. Root, "Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 412–415. [Online]. Available: http://doi.acm.org/10.1145/2591062.2591119

[21] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley Professional, 2012.

[22] J. Manweiler and R. Roy Choudhury, "Avoiding the rush hours: Wifi energy management via traffic isolation," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 253–266. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000020

[23] P. Labs, "Puppet Open Source," http://puppetlabs.com/puppet/puppet-open-source, [Online; retrieved 7-May-2014].

[24] F. Bellard, "QEMU: Open Source Processor Emulator," http://wiki.qemu.org/Main_Page, [Online; retrieved 7-May-2014].

[25] QEMU, "QEMU: Open Source Processor Emulator - Documentation/Networking," http://wiki.qemu.org/Documentation/Networking, [Online; retrieved 7-May-2014].

[26] M. McLoughlin, "The QCOW2 Image Format," https://people.gnome.org/~markmc/qcow-image-format.html, 2008, [Online; retrieved 7-May-2014].

[27] libvirt Virtualization API, "Snapshot XML Format," http://libvirt.org/formatsnapshot.html, [Online; retrieved 7-May-2014].

[28] U. Documentation, "Apt-Get," https://help.ubuntu.com/12.04/serverguide/apt-get.html, [Online; retrieved 7-May-2014].

[29] Microsoft, "Microsoft TechNet - Windows Installer Package," http://technet.microsoft.com/en-us/library/cc978328.aspx, [Online; retrieved 7-May-2014].

[30] D. Halperin, B. Greenstein, A. Sheth, and D. Wetherall, "Demystifying 802.11n power consumption," in *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924928