Software Engineering Institute
Carnegie Mellon University

# PERVASIVE MOBILE COMPUTING

*Bill Anderson, Jeff Boleng, Ben Bradshaw, James Edmondson, Grace A. Lewis, Edwin Morris, Marc Novakouski, James Root*

December 2016

## Overview

"Mobile computing" describes the use of computing technology on the go, through devices such as smartphones, tablets, portable computers, and wearable computers and sensors. The SEI is focusing on **pervasive mobile computing** at the tactical edge—environments in which front-line soldiers and first responders operate.

The SEI is working to realize this vision for soldiers and first responders:

- **Front-line soldiers:** Dismounted soldiers receive information that is more relevant and useful for their current mission, on devices that consume fewer battery and bandwidth resources, using applications that they can readily customize to support their needs.
- **Humanitarian assistance/disaster response:** On-scene responders with access to local information about resource needs of victims—such as food, water, shelter, and medicine—can make better decisions that maximize use of local resources and external assistance.

The SEI explores the architecture and implementation of mobile systems that increase the flexibility of edge users to respond to diverse missions. In these systems, "pervasive computing"—the use of devices embedded with chips that can connect to a network—plays a major role.

## Research

The SEI's research in pervasive mobile computing focuses on identifying architectures that provide flexible capability and rapid, simple deployment of capability to new generations of handheld devices, such as smartphones and wearable devices, and nearby infrastructure. The SEI is investigating tactical mobile architectures and software engineering methods and practices that can deliver the required capabilities and meet quality expectations in the dynamic and often hostile "last-mile" environment in which soldiers and first responders work.

Research projects address such critical problems as how smartphones and similar devices can best interact with infrastructure to provide optimal capability and resource conservation, how to deliver information and avoid information overload, and how to rapidly adapt smartphone apps and verify their security characteristics.

Our research in pervasive mobile computing focuses on

- **edge analytics**: mining data streams for intelligence
- **information superiority to the edge**
- **tactical cloudlets**: moving cloud computing to the edge

## Edge Analytics: Mining Data Streams for Intelligence

Warfighters and emergency first responders frequently operate in environments where personnel are required to quickly comprehend and react to rapidly-changing situations. The Edge Analytics system offers near real-time situational awareness (seconds to minutes) by analyzing social media and other high-velocity sensor data streams to provide actionable intelligence, trends, and summaries.

Our goal is to bring real-time analysis of data to tactical and emergency personnel by

- applying approximation and other strategies that allow the system to meet near real-time requirements
- leveraging contextual clues from the local environment where they are operating
- providing controls (e.g., filters) that reduce resource consumption and data volumes, increase the accuracy of analysis, and tailor the output to situational needs

The Edge Analytics System

- performs macro trend analysis (sentiment, topic, entity and location) on data slices
- analyzes social networks in real time to identify network structure and metrics
- supports interactive visualizations to allow operators to understand and digest high volumes of fast-moving data

Our research is dedicated to overcoming several challenges: large data volumes (e.g., 500 million tweets per day);time sensitivity –particularly in tactical or crisis environments;data veracity - establishing trust in data, and multi-sensor fusion –particularly open source with other forms of intelligence.

### Rapid Integration, Aggregation Filtering, and Analysis of Data for Mobile Users

Through their mobile devices, edge users may have access to numerous data sources from the military (both U.S. and allied), other government agencies, and non-government agencies, as well as from public data. If the data needed to support a mission is not available from just a single source, the ability to integrate data from multiple sources can improve the quality of the information. The SEI has developed the capability for edge users to rapidly integrate, aggregate, filter, and analyze data from multiple sources on a single mobile display.

For example, after Hurricane Sandy, first responders in New York City could have used the application to query Twitter streams for SOS messages, filtering the incoming data to contain only SOS messages from New York City. This data could then have been combined with Foursquare data to give them information about all local businesses and establishments around them. Image data from Flickr could have been added to give them live, incoming pictures of flood damage, road closures, and so on. With this capability, all of this data could have been displayed on one screen and quickly analyzed to provide better situational awareness.

## Information Superiority to the Edge

Teams of personnel operating in edge environments are often overloaded and unable to extract meaningful, useful information from the flood of data available to them. They also may not always be aware of potentially helpful information sources, or even the location and status of team members. What's more, in constrained environments, computation capabilities may be limited to what can be carried (such as a mobile phone or tablet), mobile bandwidth is at a premium, and battery power must be conserved because opportunities to recharge may be scarce. These challenges make it difficult for edge users to collect and distribute information to maintain situational awareness and to gain access to resources that are critical to achieving their defined missions.

The SEI builds mobile applications that operate with knowledge of the individual user's and the team's context to get the right information to the right people at the right time in an efficient way. So a mobile app that is aware of nearby team members can collaborate with apps carried by those users to eliminate sending and display of redundant or unnecessary data, thereby conserving both battery power and bandwidth. The app can also facilitate coordination during search and rescue operations by detecting nearby searchers and displaying their locations on a map. Context-aware apps can also provide feedback data on individual users. Many mobile devices contain position sensors, movement sensors, light sensors, and proximity sensors. The app can monitor these sensors and infer the activities of the user—for example, whether the user is running or stationary—and provide valuable status information back to leadership and the team. These features of group-context-aware apps can reduce the cognitive load on individuals and on the team by providing targeted information, and help coordinate group activities to improve the accuracy and speed with which the team completes its tasks.

The SEI seeks to improve information capture and display for warfighters and first responders using new generations of handheld devices (e.g., smartphones) at the tactical edge, for example, in Afghanistan, in Haiti after the earthquake, or in Japan after the tsunami. We do this by

- extending the use of contextual data from individual context (e.g., an individual's location) to group context, leveraging information such as missions, tasks, and roles
- making the display of information adaptive to the user's and the group's context, such as displaying friend/foe location data when warfighters are on a combat mission, or food and water availability data to first responders helping survivors of a natural disaster

- using individual and group context to optimize resources—including battery, CPU, and bandwidth—by distributing tasks performed on the group's devices in an intelligent way. For example, in a squad of 10 warfighters in close proximity, not everyone will need to use GPS for positioning, thus saving the batteries on other mobile devices with GPS turned off.
- reducing the cognitive load on warfighters. Interviews with troops who have been in the field and in combat have made clear that operating a mobile device is easy in normal conditions but difficult and rarely done while under fire. Using sensors—including vibration, audio, video, and heart rate and blood pressure monitors—mobile devices can sense environmental conditions with little or no interaction by the warfighters and first responders, allowing them to focus on the task of staying alive and completing their missions.

The architectural Ideas behind this capability are being adapted for use in advanced, human-wearable combat systems incorporating multiple biometric, environment, and situational awareness sensors and heads-up display.

### Delay-Tolerant Networking

Delay-Tolerant Networking (DTN) was originally developed for spaceflight, where communication disruption, including significant delays and packet loss, are common. DTN was adapted for environments such as mobile land networks, military ad-hoc networks, and sensor networks by Kevin Fall, who was then at Intel Research and is now CTO of the SEI. The SEI has further adapted the ideas of DTN to hand-held and wearable devices, making use of context information frequently available on those devices to make smart decisions about what data should be shared and when sharing should occur In a team environment. Among the decisions made via extensions to DTN are pre-caching of data likely to be relevant later in a mission, preferred transmission of critical data over non-critical data, prediction of the team location based on mission plans, and prioritized synchronization of information as quickly and accurately as possible when connection is reestablished.

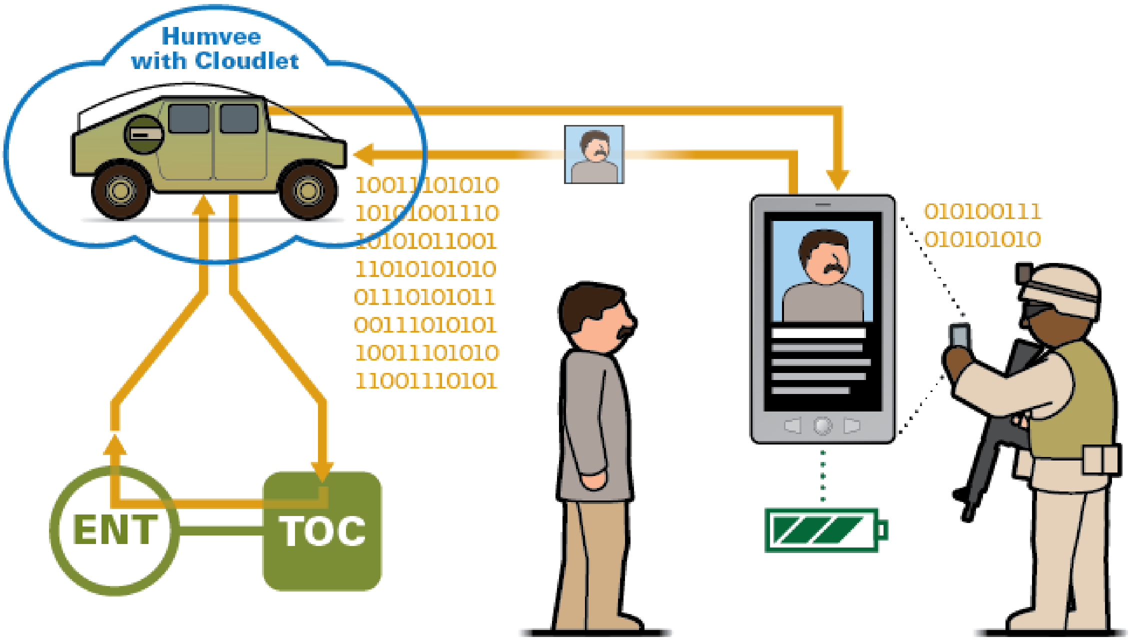
## Tactical Cloudlets: Moving Cloud Computing to the Edge

To support their missions, military and emergency personnel operating in crisis and hostile environments increasingly use mobile applications. Most of these applications perform computation-intensive tasks such as speech and image recognition, natural language processing, and situational awareness enhancement. These tasks take a heavy toll on the mobile device's battery power and computing resources. Unfortunately, battlefield and disaster environments are not only at the edge of the network infrastructure, but are also resource constrained, due to dynamic context, limited computing resources, intermittent network connectivity, and high levels of stress.

Cyber-foraging augments the capabilities of resource-limited mobile devices by leveraging compute resources in the surrounding environment. Cloudlet-based cyber-foraging relies on discoverable, generic, forward-deployed servers located in single-hop proximity of mobile devices.

Tactical Cloudlets can be hosted on vehicles or other platforms in proximity of mobile users to

- provide infrastructure to offload expensive computation
- provide forward data staging for a mission
- perform data filtering to remove unnecessary data from streams intended for dismounted users
- serve as collection points for data heading for enterprise repositories



The server's proximity allows the mobile device adequate connectivity to offload computational tasks to the cloudlet and share data and applications. Should that connectivity become lost or degraded, cloudlets can operate in disconnected mode. Moreover, they are virtual-machine based, which promotes scalability, mobility, flexibility, and elasticity.

Unlike cloudlet-based cyber-foraging, most cyber-foraging solutions rely on conventional Internet for connectivity to the cloud or strategies that tightly couple mobile clients with servers at deployment time. But these solutions don't work well in resource-constrained environments because they depend on multi-hop networks to the cloud and static deployments.

Multiple cyber-foraging systems have been developed that differ in terms of the strategy that they use to leverage remote resources—where to offload, when to offload, and what to offload. However, a review of these existing systems and research has showed that

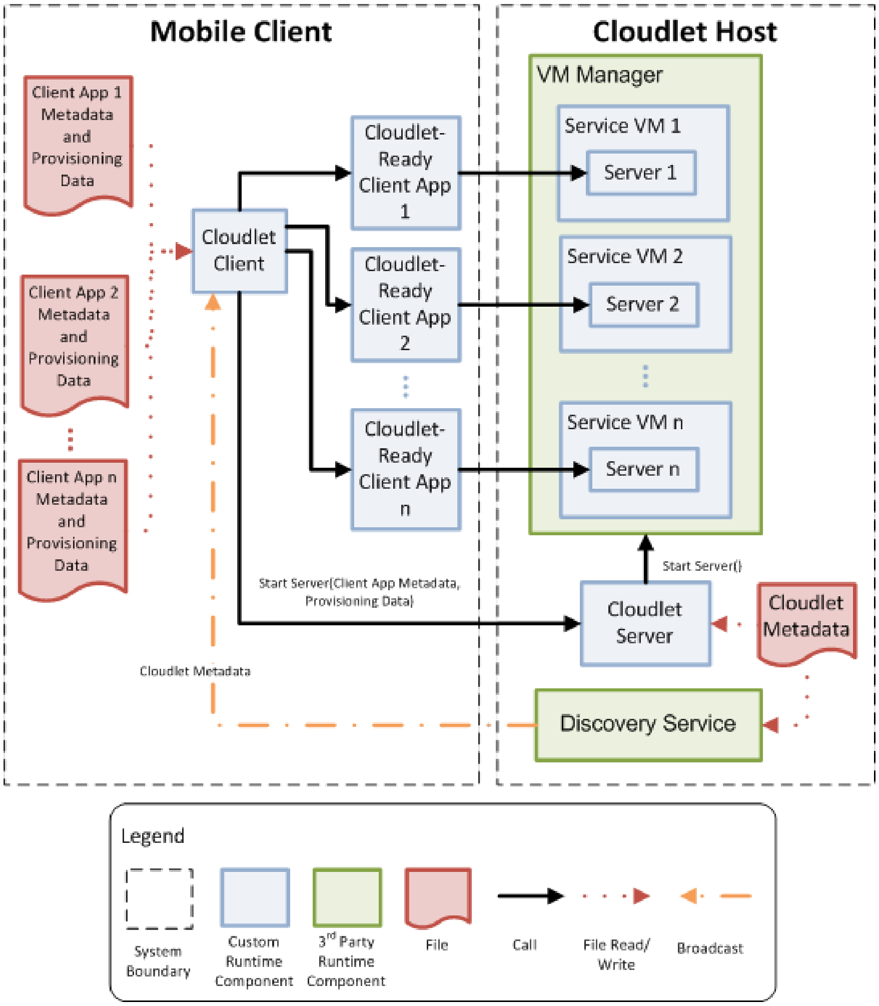
- There is emphasis on the algorithms to support code offload and state synchronization with minimal focus on software architecture and quality attributes beyond energy efficiency and performance.

- There is little guidance on how to support quality attributes such as survivability, resilience, trust and ease of deployment, critical in tactical environments.

## Tactical Cloudlets: Research Focus

Our research focuses on developing cyber-foraging architectures and implementations that address tactical system quality attributes and therefore support deployment of tactical cloudlets as part of mobile tactical systems.

Based on the reference architecture shown below, we implemented five different cloudlet provisioning mechanisms that range from very dynamic mechanisms in which cloudlets are provisioned from mobile devices to more static provisioning strategies in which cloudlets are pre-provisioned with capabilities based on mission requirements.


Mobile Client
Cloudlet Host
Client App 1 Metadata and Provisioning Data
Client App 2 Metadata and Provisioning Data
Client App n Metadata and Provisioning Data
Cloudlet Client
Cloudlet-Ready Client App 1
Cloudlet-Ready Client App 2
Cloudlet-Ready Client App n
VM Manager
Service VM 1
Server 1
Service VM 2
Server 2
Service VM n
Server n
Start Server()
Start Server(Client App Metadata, Provisioning Data)
Cloudlet Server
Cloudlet Metadata
Cloudlet Metadata
Discovery Service
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Call
File Read/ Write
Broadcast

The results of experiments based on the different cloudlet provisioning mechanisms led us to combine Cached VM with Cloudlet Push as the cloudlet provisioning mechanism to enable lower energy consumption on the mobile device, place less requirements on mobile devices, and simplify cloudlet provisioning in tactical environments.

In our goal to develop architectural strategies and prototypes for cyber-foraging that consider a wider range of critical quality attributes not considered by the commercial mobile ecosystem, our current focus is on survivability, in particular the survivability of mobile tactical systems. Our hypothesis is that survivability for tactical mobile systems requires quick deployment and redeployment of capabilities, fast execution times so that computation can take place before cloudlets become out of range, and support for disconnected operations (between cloudlets and data centers).

These are features of our baseline tactical cloudlets implementation that contribute to increased survivability of tactical mobile systems:

- Capabilities as Services: Based on the definition of a service in the context of service orientation, each VM provides a self-contained capability and exposes a simple interface. Metadata in the form of keywords is used by the cloudlet discovery protocol to inform mobile devices of available capabilities.

- Request-Response Nature of Interactions Between Mobile Devices and Cloudlets: In the case of computation offload, tactical cloudlets are best fit for applications based on stateless, request-response, client/server interactions.This type of interaction enables easy detection of failed communication between mobile devices and cloudlets as well as minimal effect on mobile devices if computation needs to be restarted or migrated.

- Virtual machines as service containers: VMs can be started and stopped asneeded based on number of active users (which is typically bounded in edge environments because group size is known) therefore supporting scalability and elasticity.

- Cloudlet Management Component: A lightweight, web-based interface to the Cloudlet Server and Service VM repository enables easy deployment and redeployment of capabilities.

- Standard packaging of Service VMs: Service VMs can be easily installed from the cloudlet manager, an enterprise Service VM repository, a thumb drive, or the mobile device connected via USB to the cloudlet.

- Optimal cloudlet selection: We extended the cloudlet discovery protocol to use metadata from the client app, Service VM, and the cloudlet so that in the case that there is more than one cloudlet in range, the mobile device can automatically select the cloudlet that maximizes a pluggable utility function. This function can be based on cloudlet load, signal strength, or any other parameter.

- Manual and automated cloudlet handoff: We are adding VM migration capabilities to enable manual and automated handoff of data and computation between cloudlets that are within range of each other. Manual handoff would enable scenarios in which a user is migrating capabilities from a fixed cloudlet to a mobile cloudlet to support field operations, as well as reintegration back to the fixed cloudlet. Automated migration would enable load balancing, similar to what is done in cloud data centers for resource optimization.

### Cloudlet-Based Cyber-Foraging to Optimize Resource Consumption

The goal of this project is to optimize resources and increase computation capability of mobile devices by using cloudlets as code-offload elements. Cloudlets are discoverable, localized, stateless servers running one or more virtual machines on which soldiers can offload expensive computations from their handheld mobile devices. Cloudlets enhance processing capacity and conserve battery power while providing ease of deployment in the field.

We define a three-level architecture in which cloudlets are intermediate offload elements that sit between mobile devices and the enterprise cloud. In this architecture, cloudlets are located close to the mobile devices that they serve, for example, in a Tactical Operations Center, vehicle on the ground, or unmanned aerial vehicle flying overhead. This proximity decreases latency and improves network resiliency by using a single-hop network, and it potentially lowers battery consumption by using WiFi or short-range instead of broadband wireless, which typically consumes more energy.

A key attribute of this architecture is that cloudlets are stateless. Communication between the cloudlet and the enterprise cloud is required only during provisioning. Once a cloudlet is provisioned, its connection to the enterprise cloud can be disrupted without affecting offload service to mobile clients. Adding a new offload element or replacing an existing one involves very little setup or configuration effort. In addition, the virtual-machine technology in cloudlets provides greater flexibility in the type and platform of applications and reduces setup and administration time, which is critical for systems at the tactical edge.

### Cloudlet Provisioning Mechanisms

A key aspect of cloudlet-based cyber-foraging is cloudlet provisioning—configuring and deploying the *Service VM* that contains the server code (i.e., server portion of the application) on the cloudlet so that it is ready to use by the client running on the mobile device.

We have implemented five cloudlet provisioning mechanisms: Optimized VM Synthesis, Application Virtualization, Cached VM, Cloudlet Push, and On-Demand VM Provisioning.

The cloudlet provisioning mechanisms share a set of common components.

- The Mobile Client is an Android-based smartphone.
- The Cloudlet Client and Cloudlet-Ready Apps are implemented as Android apps (Java).
- The VM Manager is QEMU-KVM. KVM arbitrates access to the CPU and memory and QEMU creates and manages VMs using the KVM kernel module [KVM 2014, QEMU 2014].
- The API used to interact with QEMU is libvirt [Libvirt 2014].
- The qemu-img tool is a QEMU disk image utility [Bellard 2013].

- Each Service VM is a guest VM in QEMU-KVM that runs the Application Server that corresponds to the server portion of the Cloudlet-Ready App.
- The Cloudlet Server is implemented as an HTTP Server using CherryPy for most mechanisms, except for Application Virtualization, which uses Jetty [CherryPy 2014, Eclipse 2014].
- The Discovery Service is a Zeroconf-based implementation [Zeroconf 2014].
- Port Forwarding (NAT) (for Network Address Translation) is used to forward packets received by the Cloudlet Host to the port on which each Service VM is listening.

Each cloudlet provisioning mechanism will include descriptions of components that are specific to the implementation of the mechanism.

### Cached VM

In Cached VM, the cloudlet is pre-provisioned with Service VMs that correspond to capabilities that match the client apps on the mobile device. The high-level architecture for cloudlet provisioning using Cached VM is shown in the figure below. Details specific to this provisioning mechanism are as follows.

Each *Service VM* has a unique service identifier and is stored in the *Service VM Repository* as a set of files:

- Service VM Metadata file (.jsonsvm): JSON* file with the following fields:
  - serviceId: unique identifier for the service provided by the VM
  - servicePort: Port on which the server inside the VM will be listening for connections

  **JSON stands for JavaScript Object notation and is a widely used lightweight data interchange format*.

- Disk Image file (.qcow2): qcow2 file that contain the VM disk image and contains the server portion of the application

- VM State Image file (.lqs): VM state image in the format that libvirt.save() generates when saving a memory image. This format is called Libvirt Qemu Saved (lqs), and includes both the description of the VM that was suspended, as well as the memory state of the VM that includes the running server [Libvirt 2014].

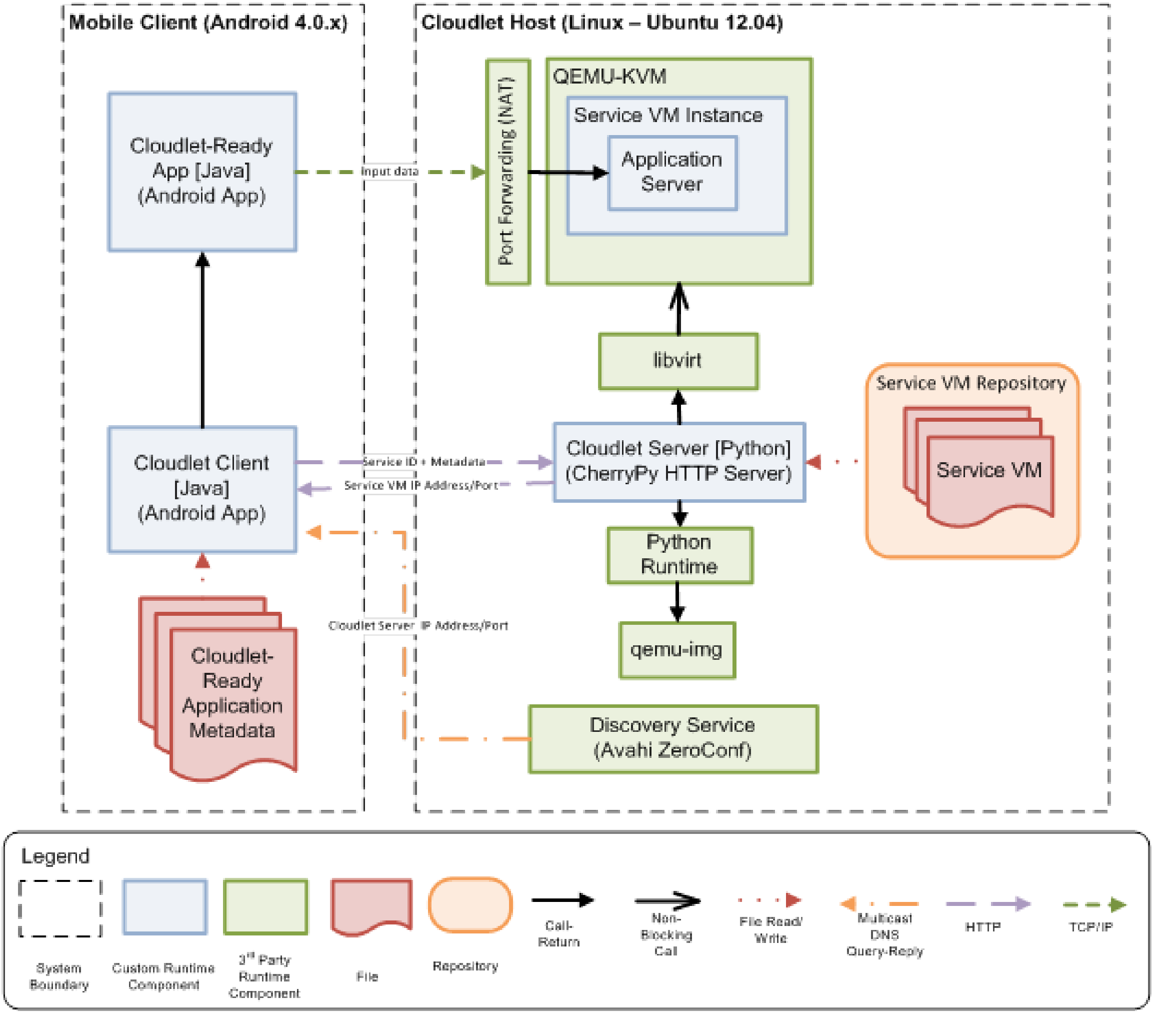
The *Service VM Repository* is set up as a set of folders, each of which uses the following naming convention:

- <serviceId>

  <serviceId>.jsonsvm

  <serviceId>.qcow2

  <serviceId>.qcow2.lqs

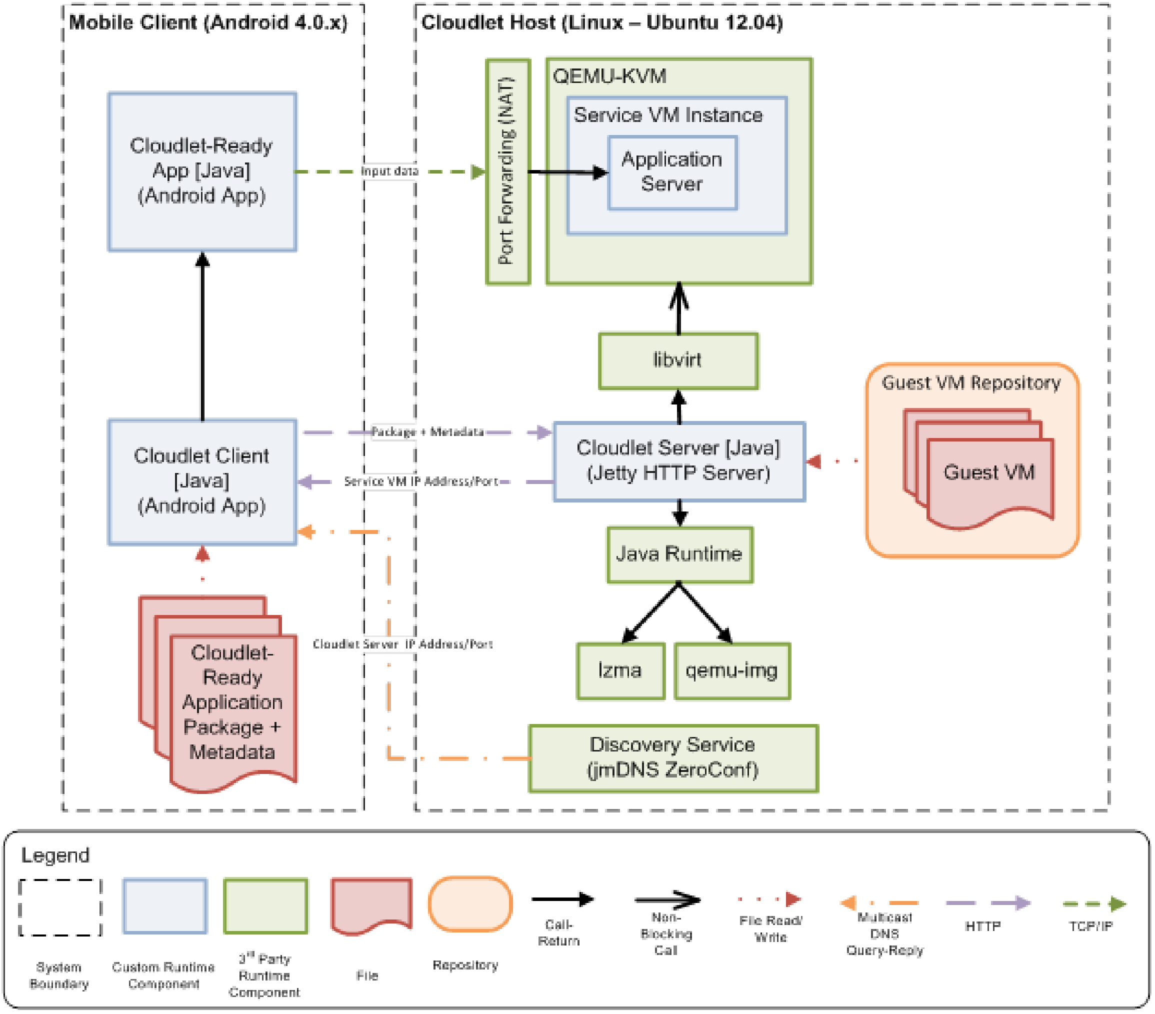

Mobile Client (Android 4.0.x)
Cloudlet Host (Linux – Ubuntu 12.04)
Cloudlet-Ready App [Java] (Android App)
Input data
Port Forwarding (NAT)
QEMU-KVM
Service VM Instance
Application Server
libvirt
Service VM Repository
Service VM
Cloudlet Client [Java] (Android App)
Service ID + Metadata
Service VM IP Address/Port
Cloudlet Server [Python] (CherryPy HTTP Server)
Python Runtime
qemu-img
Cloudlet Server IP Address/Port
Cloudlet-Ready Application Metadata
Discovery Service (Avahi ZeroConf)
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Repository
Call-Return
Non-Blocking Call
File Read/ Write
Multicast DNS Query-Reply
HTTP
TCP/IP

A *Service VM* is created by installing the application server inside a VM and then suspending it. The JSON file is created automatically by the script that we use to create the *Service VM* and is stored in the same directory as the disk and memory image files. These three files are used to launch a *Service VM* so that it starts from the point at which it was suspended, therefore reducing application-ready time.

## Application Virtualization

In Application Virtualization the cloudlet is also provisioned from the mobile device at runtime. Application Virtualization uses an approach similar to operating system (OS) virtualization, by "tricking" the software into interacting with a virtual rather than actual environment. A runtime component intercepts all system calls from an application and redirects these to resources inside the virtualized application. Virtualized applications are created in advance for server portions of applications using tools that package the application with all its dependencies. CDE (short for Code, Data and Environment) was used as the application virtualizer for Linux [Guo 2011] and Cameyo was used for Windows [Cameyo 2013]. CDE virtualizes applications by monitoring their execution and Cameyo by monitoring their installation process. Both tools produce virtualized applications that are loaded on the mobile device and at runtime are sent to the cloudlet to be deployed in a VM that matches the OS of the virtualized application. A full implementation is described, analyzed, and compared to VM synthesis by Messinger [Messinger 2013]. The high-level architecture for cloudlet provisioning using application virtualization is shown in the figure below. Details specific to this provisioning mechanism are below:

- The *Cloudlet-Ready Application Package* is the virtualized application that corresponds to the server portion of the *Cloudlet-Ready App* and is created using the process just described.
- The *Guest VM Repository* contains VM disk image files for each operating system supported by the cloudlet.


Mobile Client (Android 4.0.x)
Cloudlet Host (Linux – Ubuntu 12.04)
Cloudlet-Ready App [Java] (Android App)
Input data
Port Forwarding (NAT)
QEMU-KVM
Service VM Instance
Application Server
libvirt
Guest VM Repository
Guest VM
Package + Metadata
Service VM IP Address/Port
Cloudlet Client [Java] (Android App)
Cloudlet Server [Java] (Jetty HTTP Server)
Java Runtime
Cloudlet-Ready Application Package + Metadata
Cloudlet Server IP Address/Port
lzma
qemu-img
Discovery Service (jmDNS ZeroConf)
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Repository
Call-Return
Non-Blocking Call
File Read/ Write
Multicast DNS Query-Reply
HTTP
TCP/IP

High-Level Architecture for Application Virtualization

## Optimized VM Synthesis

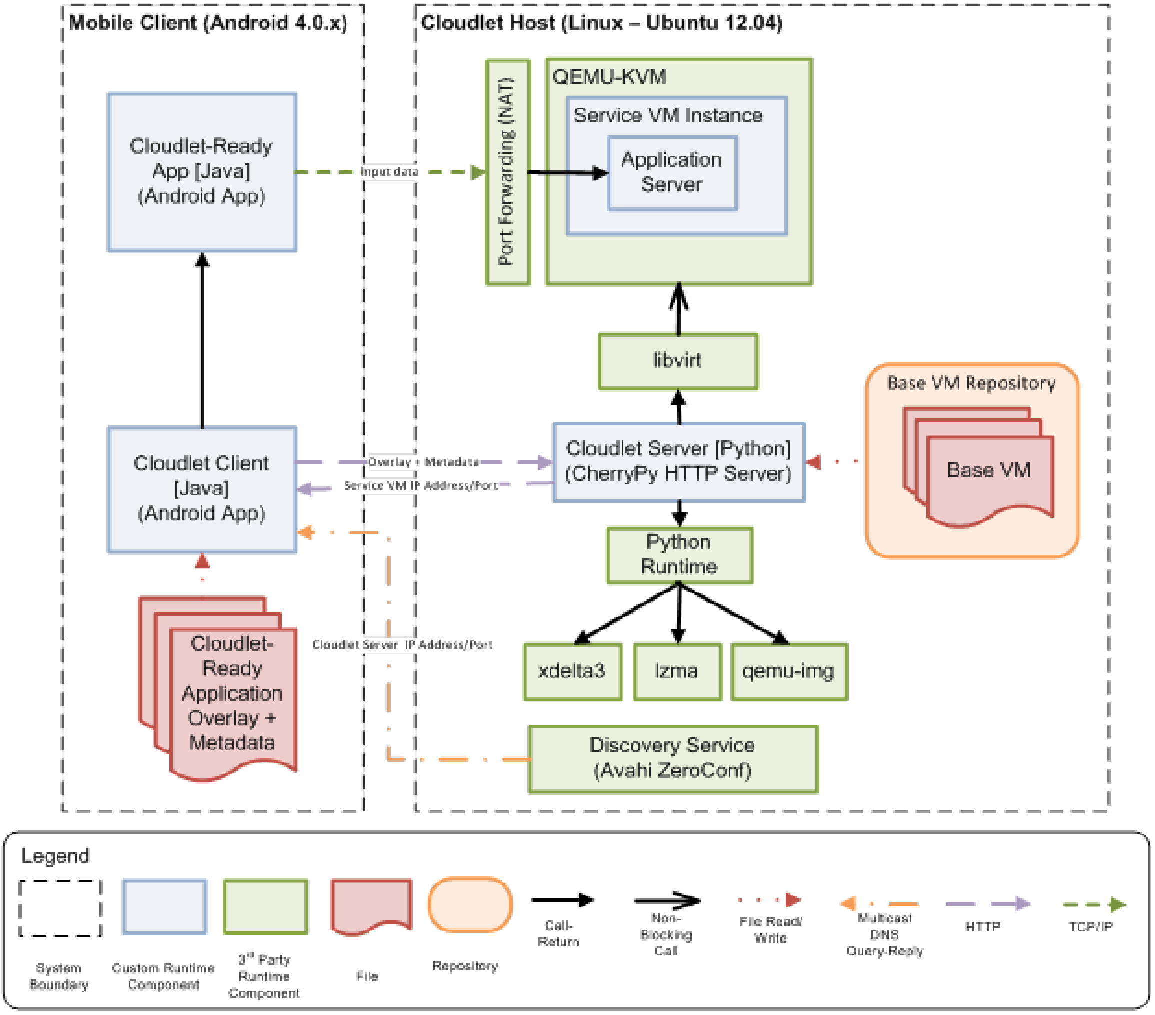
In VM synthesis, the cloudlet is provisioned from the mobile device at runtime. The original implementation of VM synthesis is fully described by Simanta and colleagues [Simanta 2013].

In the Optimized VM Synthesis implementation, the goal is to reduce application-ready time – the time between the cloudlet provisioning request and the notification that the server is ready for execution. The high-level architecture for cloudlet provisioning using Optimized VM Synthesis is shown in Figure 1. Details specific to this provisioning mechanism are below:

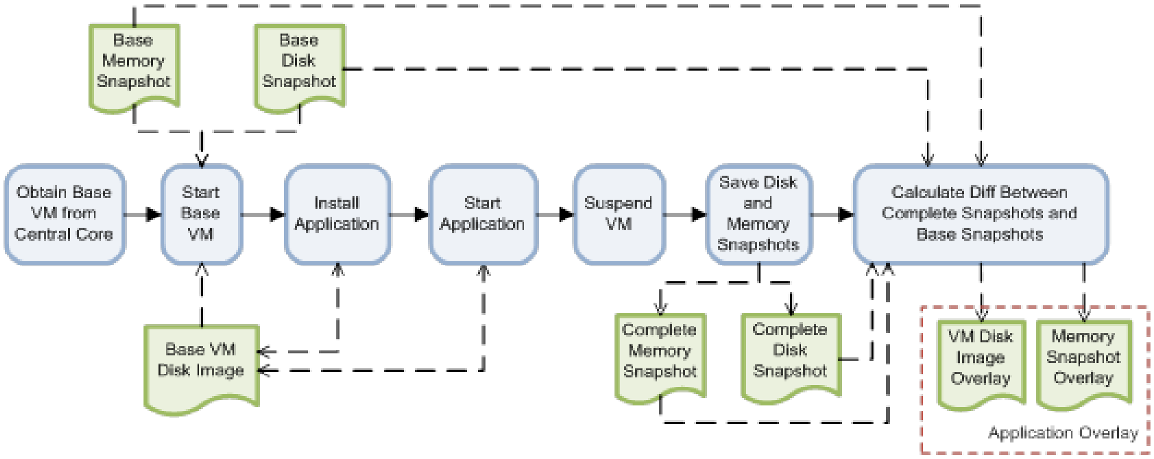
- An *Application Overlay* that corresponds to the server portion of a client-server application is created once offline, by starting a VM instance from a Base VM disk image file (that uses QEMU copy on write 2 (qcow2) [McLoughlin 2008] as the VM disk image file format) and a base memory image, installing the server on the base VM image, and suspending the VM. When VM is suspended, two files are created as part of the application overlay: one that corresponds to the disk image differences between the suspended VM and the base VM (the qcow2 file) and another that corresponds to the binary difference between the suspended memory image and the base memory image (calculated using xdelta3 [Xdelta 2014] and VCDIFF [JmDNS 2011] format). These calculated overlays (disk and memory) are compressed using LZMA2 with the XZ stream compression format [Lehr 2002] and loaded on the mobile device. The overlay creation process is shown in Figure 2.

- The *Base VM Repository* contains the set of all *Base VM* disk image and memory image files from which application overlays were created.

- The *xdelta3* [Xdelta 2014] and lzma [Python 2014] libraries are used by the *Cloudlet Server* at runtime to decompress and apply the received application overlay to its corresponding *Base VM* (details in Figure 2).


Mobile Client (Android 4.0.x)
Cloudlet Host (Linux – Ubuntu 12.04)
Cloudlet-Ready App [Java] (Android App)
Input data
Port Forwarding (NAT)
QEMU-KVM
Service VM Instance
Application Server
libvirt
Base VM Repository
Base VM
Cloudlet Client [Java] (Android App)
Overlay + Metadata
Service VM IP Address/Port
Cloudlet Server [Python] (CherryPy HTTP Server)
Python Runtime
Cloudlet-Ready Application Overlay + Metadata
Cloudlet Server IP Address/Port
xdelta3
lzma
qemu-img
Discovery Service (Avahi ZeroConf)
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Repository
Call-Return
Non-Blocking Call
File Read/ Write
Multicast DNS Query-Reply
HTTP
TCP/IP

High-Level Architecture for VM Synthesis



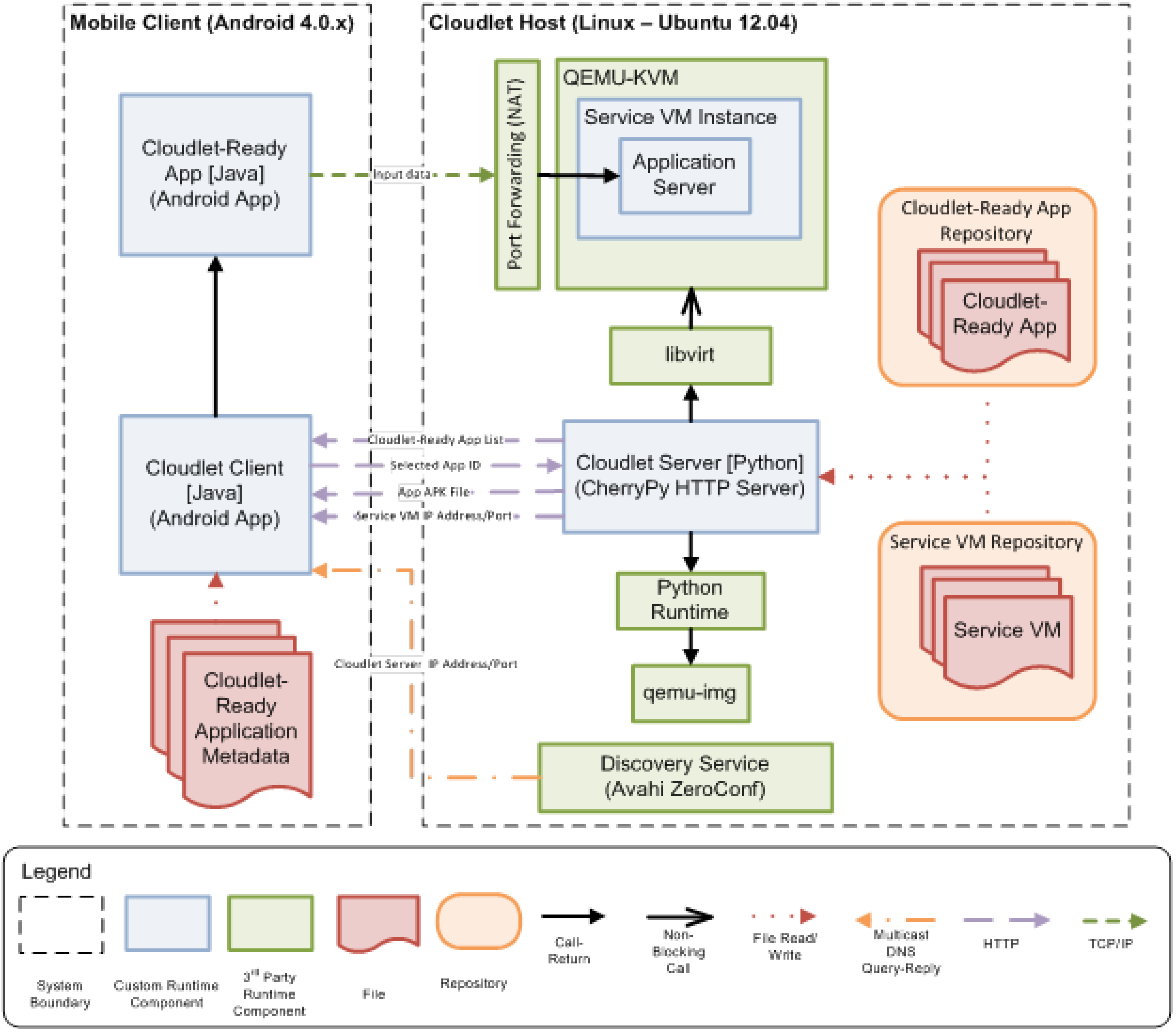
Application Overlay Creation Process for VM Synthesis

## Cloudlet Push

In Cloudlet Push, the cloudlet is not only pre-provisioned with Service VMs, but also mobile client apps that can use the Service VMs. The high-level architecture for cloudlet provisioning using Cloudlet Push is shown in the figure below. Details specific to this provisioning mechanism are as follows:

- Each *Cloudlet-Ready App* has a unique identifier and additional metadata that is stored in the *Cloudlet-Ready App Repository*. The *Cloudlet Server* uses this information to build a *Cloudlet-Ready App List* that is sent to the *Cloudlet Client* upon request, similar to accessing an app store. It is important to note that the only element that is needed on the *Mobile Client* is the *Cloudlet Client*.

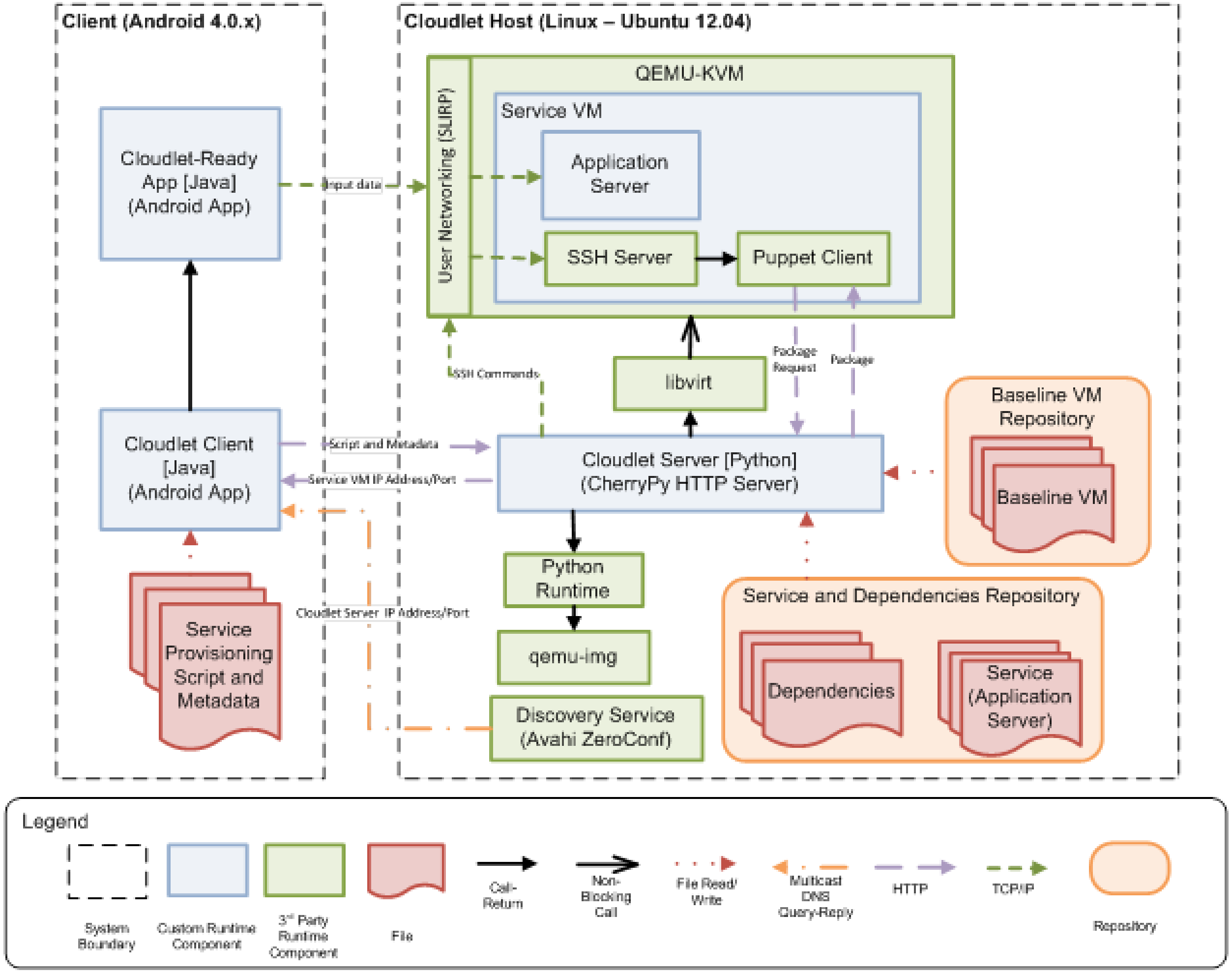
- The *Cloudlet-Ready App* and its *Cloudlet-Ready Application Metadata* are obtained from the cloudlet at runtime and saved on the *Mobile Client*.


Mobile Client (Android 4.0.x)
Cloudlet Host (Linux – Ubuntu 12.04)
Cloudlet-Ready App [Java] (Android App)
Input data
Port Forwarding (NAT)
QEMU-KVM
Service VM Instance
Application Server
Cloudlet-Ready App Repository
Cloudlet-Ready App
libvirt
Cloudlet Client [Java] (Android App)
Cloudlet-Ready App List
Selected App ID
App APK File
Service VM IP Address/Port
Cloudlet Server [Python] (CherryPy HTTP Server)
Service VM Repository
Service VM
Python Runtime
qemu-img
Cloudlet-Ready Application Metadata
Cloudlet Server IP Address/Port
Discovery Service (Avahi ZeroConf)
Legend
System Boundary
Custom Runtime Component
3rd Party Runtime Component
File
Repository
Call-Return
Non-Blocking Call
File Read/ Write
Multicast DNS Query-Reply
HTTP
TCP/IP

High-Level-Architecture for Cloudlet Push

### On-Demand VM Provisioning

In On-Demand VM Provisioning, a commercial cloud provisioning tool is used to assemble a *Service VM*. In this case the cloudlet has access to all the elements for putting together a *Service VM* based on a provisioning script. The experimental prototype uses Puppet, and the provisioning script is a manifest that is written in Puppet's declarative language [Puppet 2014]. The high-level architecture for cloudlet provisioning using On-Demand VM Provisioning is shown in the figure. Below the figure are details specific to this provisioning mechanism.


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

High-Level Architecture for On-Demand Provisioning

A *Baseline VM* is a suspended VM that is used as a template for the construction of Service VMs. It is important to note that unlike Base VMs in VM Synthesis, *Baseline VMs* can be modified, updated, and maintained continuously without affecting the provisioning process. Each *Baseline VM* is stored in the *Baseline VM Repository* as a set of files:

- Disk Image file (.qcow2): qcow2 file that corresponds to the VM disk image of the *Baseline VM*. It should contain a basic OS installation plus any components and libraries commonly used by *Service VMs*. In the experimental prototype that uses Puppet, the following components need to be part of the Baseline VM:
    - SSH Server for the *Cloudlet Server* to send files and commands. In this provisioning mechanism, *QEMU-KVM* is set up with *User Networking* [QEMU 2014] so that the *Cloudlet Server* can send SSH commands to the *Service VM*.
    - Puppet Client for the execution of Puppet manifests inside the VM.

    - Link to the *Cloudlet Server* so that the *Service VM* can download packages and dependencies that are stored locally on the Cloudlet (via HTTP with the *Cloudlet Serve*r acting as an HTTP file server).

- VM State Image file (.lqs): VM state image in the format that libvirt.save() generates when saving a memory image
- Baseline VM Metadata file (.jsonbmd): JSON file with the following fields that describes the basic features of the Baseline VM
    - osFamily (string): Basic OS family (e.g., “Linux”, “Windows”)
    - os (string): Name of the OS or distribution (e.g., “Windows 7”, “Ubuntu”)
    - osVersion (string): OS version (e.g., “SP1”, “8.1”, “12.10”)
    - osISA: Instruction set (e.g., “x86-32” or “x86-64”)

The *Baseline VM Repository* is set up as a set of folders, each of which uses the following naming convention:

- <baseline-vm-id>
    - <baseline-vm-id>.jsonbmd
    - <baseline-vm-id>.qcow2
    - <baseline-vm-id>.qcow2.lqs <

Each Cloudlet-Ready App has a corresponding Service Provisioning Script and Metadata that are used to set up the corresponding Service VM on the cloudlet.

- Baseline VM Metadata file (.jsonbmd): JSON file that describes the basic features of the Baseline VM that is needed to create the Service VM. The format is the same as the Baseline VM Metadata file described above and is used during the provisioning process to find matching Baseline VMs.
- Service VM Metadata file (.jsonsvm): JSON file with the same structure defined in the explanation of Cached VM to describe a Service VM.
- Puppet Manifest: Script with instructions on what must be provisioned inside the Service VM, such as the application server to install, dependencies, and libraries. The script follows the standard for Puppet manifests [Puppet Labs 2014].

The *Services and Dependencies Repository* contains two sets of files. Although in the experimental prototype these files are retrieved from the *Cloudlet Host* using HTTP download, they could also be obtained from an external URL if there were connectivity.

- Files that compose the actual service to be provided by the Service VM (implemented in the Application Server)
- *Dependencies*: Files that compose any dependencies/libraries that could be used by Services. In the experimental prototype it is implemented for Ubuntu as an APT-GET repository and for Windows as MSI packages.

## Tactical Cloudlets: Future Research

The next quality attribute that we will focus on in our research is trust. Our current cloudlet implementation relies on the security provided by the network, that is, a mobile device is allowed to interact with a cloudlet according to network policies and permissions. This means that the cloudlet implementation is as secure as the network. While this may be acceptable in many domains, it is likely not enough for tactical environments.

A key aspect of cloudlets is that they are discoverable. The Cloudlet Client that is installed on a cloudlet-enabled mobile device uses Multicast DNS to query for cloudlets (set up as cloudlet services by the Discovery Service that runs on the cloudlet). Multicast DNS protocols are known to be insecure. However, securing the discovery process is not the problem because port scans or other probing methods can easily bypass discovery.

A potential starting point for embedding security in our cloudlet implementation is establishing that initial trust between mobile devices and cloudlets; that is

- As a mobile device, is what I discovered really a "friendly" cloudlet?
- As a cloudlet, did that offloading request really come from a "friendly" mobile device?

A common solution for establishing trust between two nodes is to use a third-party, online trusted authority that validates the credentials of the requester or a certificate repository. However, the characteristics of tactical edge environments do not consistently provide access to that third-party authority or certificate repository because they are DIL environments (disconnected, intermittent, limited).

Our future research will explore solutions for establishing trusted identities in disconnected environments. Even though the motivation comes from cloudlets, the goal is for the results to be applied to any form of trusted communication betwen two or more computing nodes. A review of related work shows that this is indeed a challenge and there are many relevant and interesting ideas but not very many specific solutions.

# Bibliography

## Pervasive Computing

### [Morris & Elm 2014]

Morris, Edwin J. & Elm, Joseph P. *Advanced Mobile Systems Initiative 2014*. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=87954

## Edge Analytics

### [Anderson & Boleng 2014]

Anderson, William & Boleng, Jeff. *Edge Analytics - Real-time Analysis of Social Media* Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=77189

### [Simanta & Anderson 2013]

Simanta, Soumya & Anderson, William. *Edge Analytics - Real-Time Analysis of High-Volume Streaming Data.* Software Engineering Institute, Carnegie Mellon University. 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=87575

## Information Superiority to the Edge

### [Lewis et al 2013]

Lewis, Grace; Boleng, Jeff; Cahill, Gene; Morris, Edwin J.; Novakouski, Marc; Root, James; & Simanta, Soumya. *Architecture Patterns for Mobile Systems in Resource-Constrained Environments (SATURN 2013)*. Software Engineering Institute, Carnegie Mellon University. 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=48535

### [Lewis et al 2012a]

Lewis, Grace; Novakouski, Marc; & Sanchez, Enrique. *A Reference Architecture for Group-Context-Aware Mobile Applications.* Software Engineering Institute, Carnegie Mellon University. 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=435796

### [Lewis et al 2012b]

Lewis, Grace; Novakouski, Marc; & Sanchez, Enrique. *Extensibility as a Collaboration Enabler: A Case Study for Group-Context-Aware Mobile Applications (SATURN 2012)*. Software Engineering Institute, Carnegie Mellon University. 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=20898

**[Novakouski 2012]**

Novakouski, Marc. Group-Context-Aware Mobile Applications [blog post]. *SEI Insights.* May 21, 2012. https://insights.sei.cmu.edu/sei_blog/2012/05/group-context-aware-mobile-applications.html

**[Simanta & Miller 2013]**

Simanta, Soumya & Miller, Suzanne (interviewer). Situational Awareness Mashups. *SEI Podcast Series.* November 14, 2013. http://www.sei.cmu.edu/podcasts/podcast_episode.cfm?episodeid=294287

**[Yan and Lewis 2014]**

Yan, Bryan; & Lewis, Grace. *Evaluation of the Applicability of HTML5 for Mobile Applications in Resource-Constrained Edge Environments.* CMU/SEI-2014-TN-002. Software Engineering Institute, Carnegie Mellon University. 2014. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=295973

## Tactical Cloudlets

**[Cameyo 2013]**

*Cameyo Website.* (2013). http://www.cameyo.com/

**[CherryPy 2014]**

CherryPy: A Minimalist Python Web Framework. *CherryPy Website*. 2014. / http://www.cherrypy.org/

**[Eclipse 2014]**

Jetty - Servlet Engine and Http Server. *Eclipse Website.* 2014. http://www.eclipse.org/jetty/

**[Guo & Engler 2011]**

Guo, Phillip J. & Engler, Dawson. CDE: Using system call interposition to automatically create portable software packages. *Proceedings of the 2011 USENIX Annual Technical Conference* (pp. 247-252), 2011. https://www.usenix.org/legacy/events/atc11/tech/final_files/GuoEngler.pdf

**[Ha et al 2011]**

Ha, Kiryong; Lewis, Grace, Simanta, Soumya; & Satyanarayanan, Mahadev. *Cloud Offload in Hostile Environments* (CMU-CS-11-146). School of Computer Science, Carnegie Mellon University, 2011. https://www.cs.cmu.edu/~satya/docdir/CMU-CS-11-146.pdf

**[KVM 2014]**

KVM. Kernel Based Virtual Machine. *Linux KVM Website*. 2014. http://www.linux-kvm.org/

### [Lewis & Miller 2014]

Lewis, Grace & Miller, Suzanne (interviewer). Tactical Cloudlets. *SEI Podcast Series.* December 4, 2014. http://csauth-sei.sei.cmu.edu/podcasts/podcast_episode.cfm?episodeid=428908

### [Lewis 2013]

Lewis, Grace. Tactical Cloudlets: Moving Cloud Computing to the Edge [blog post]. *SEI Insights.* Nivember 10, 2014. http://insights.sei.cmu.edu/sei_blog/2014/11/tactical-cloudlets-moving-cloud-computing-to-the-edge.html

### [Libvert 2014]

Libvert: The Virtualization API. *Libvirt Virtualization API Website.* 2014. http://libvirt.org

### [Messinger & Lewis 2013]

Messinger, D. & Lewis, G. *Application Virtualization as a Strategy for Cyber Foraging in Resource-Constrained Environments* (CMU/SEI-2013-TN-007). Software Engineering Institute, Carnegie Mellon University. 2013. http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=53234

### [Puppet 2014]

Learning Puppet—Manifests. *Puppet Website.* 2014. http://docs.puppetlabs.com/learning/manifests.html

### [QEMU 2014]

QEMU Open Source Processor Emulator. *QEMU Wiki.* 2014. http://wiki.qemu.org

### [Simanta et al 2013]

Simanta, Soumya; Ha, Kiryong; Lewis, Grace; Morris, Edwin; & Satyanarayanan, Mahadev. “A Reference Architecture for Mobile Code Offload in Hostile Environments.” Presented at the 4th International Conference on Mobile Computing, Applications and Services (MobiCASE 2012), Seattle, WA, Oct. 2012. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering,* 110 (2013): 274–193. http://www.cs.cmu.edu/~satya/docdir/simanta-mobicase2012.pdf

### [Simanta et al 2012]

Simanta, Soumya; Lewis, Grace; Morris, Edwin; Ha, Kiryong; & Satyanarayanan, Mahadev. *Cloud Computing at the Tactical Edge* (CMU/SEI-2012-TN-015). Software Engineering Institute, Carnegie Mellon University, 2012. http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=28021

### [Zeroconf 2014]

Zero Configuration Networking (Zeroconf). *Zeroconf Website.* 2014. http://www.zeroconf.org/

# Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

**Phone**: 412/268.5800 | 888.201.4479
**Web**: www.sei.cmu.edu | www.cert.org
**Email**: info@sei.cmu.edu

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.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS” BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

DM-0004470