Software Engineering Institute
Carnegie Mellon University

Cyber-Foraging for Improving Survivability of Mobile Systems

Sebastián Echeverría (Universidad de los Andes)
Grace A. Lewis
James Root
Ben Bradshaw

February 2016

TECHNICAL REPORT
CMU/SEI-2016-TR-001

Software Solutions Division (SSD)

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

http://www.sei.cmu.edu

Copyright 2016 Carnegie Mellon University

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.

This report was prepared for the

SEI Administrative Agent
AFLCMC/PZM
20 Schilling Circle, Bldg. 1305, 3rd floor
Hanscom AFB, MA 01731-2125

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.

[Distribution Statement A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

Internal use:* Permission to reproduce this material and to prepare derivative works from this material for internal use is granted, provided the copyright and “No Warranty” statements are included with all reproductions and derivative works.

External use:* This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other external and/or commercial use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

* These restrictions do not apply to U.S. government entities.

DM-0003259

Table of Contents

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

List of Figures

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

List of Tables

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Abstract

Cyber-foraging is a technique for dynamically augmenting the computing power of resource-limited mobile devices by opportunistically exploiting nearby fixed computing infrastructure. Cloudlet-based cyber-foraging relies on discoverable, generic, forward-deployed servers located in single-hop proximity of mobile devices. We define *tactical cloudlets* as the infrastructure to support computation offload and data staging at the tactical edge. However, the characteristics of tactical environments—such as dynamic context, limited computing resources, disconnected-intermittent-limited (DIL) network connectivity, and high levels of stress—pose a challenge for the continued operations of mobile systems that leverage cloudlets in tactical environments. We also define *survivability of mobile systems* as the capability of a system to continue functioning in spite of adversity. This report presents an architecture and experimental results that demonstrate that cyber-foraging using tactical cloudlets increases the survivability of mobile systems.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

1 Introduction

Cyber-foraging is an area of work within mobile cloud computing that leverages external resources (i.e., cloud servers or local servers called surrogates) to augment the computation and storage capabilities of resource-limited mobile devices while extending their battery life [Satyanarayanan 2001]. There are two main forms of cyber-foraging. One is computation offload, which is the offloading of expensive computation to extend battery life and increase computational capability. The second is data staging, which involves temporarily staging data in transit on intermediate surrogates to improve data transfers between mobile devices and the cloud.

Cloudlet-based cyber-foraging relies on discoverable, generic, forward-deployed servers that are located in single-hop proximity of mobile devices [Satyanarayanan 2009]. We further define *tactical cloudlets* as cloudlets that support computation offload and data staging at the tactical edge [Lewis 2014a]. However, the characteristics of tactical environments, such as dynamic context, limited computing resources, disconnected-intermittent-limited (DIL) network connectivity, and high levels of stress, pose a challenge for the continued operations of mobile systems that leverage cloudlets in tactical environments.

Battery savings and better response times (lower latency) are demonstrated benefits of cyber-foraging and contributors to mission success [Lewis 2014b]. However, mission success also requires cloudlet-deployed capabilities to be available when they are required, despite the challenges of tactical environments.

This report presents our implementation of tactical cloudlets with a specific focus on survivability, defined as the capability of a system to continue functioning in spite of adversity. Section 2 presents a summary of related work in cyber-foraging and software systems survivability. Section 3 presents the architecture of our tactical cloudlet implementation. Section 4 focuses on the tactical cloudlet features that promote survivability of mobile systems in the field. Finally, Section 5 concludes the report and outlines next steps.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

2 Related Work in Cyber-Foraging and Software Systems Survivability

2.1 Cyber-Foraging

We recently conducted a systematic literature review (SLR) on architectures for cyber-foraging systems that identified 57 primary studies and 60 cyber-foraging systems. 52 of these systems were computation offload systems, and 8 were data staging systems [Lewis 2014a]. The SLR showed that there is indeed active work in this area but also that there is (1) a lack of understanding of system qualities attributes beyond energy, performance, network usage, and fidelity of results, and (2) a lack of focus on system-level concerns (e.g., ease of distribution and installation, survivability and security) that are required when moving from experimental prototypes to operational systems. The goal of the work described in this report is to fill in some of these gaps.

As far as the use of cyber-foraging and cloudlets in tactical environments, there is indeed motivation for their use as outlined in a report titled *The Role of Cloudlets in Hostile Environments*. [Satyanarayanan 2013]. In that report, the authors propose using proximate, VM-based cloudlets as a way to overcome the lack of a reliable, high-bandwidth, end-to-end network, which is difficult to guarantee in hostile environments. More specifically, there is recent work in deploying cloudlet-like servers at the tactical edge to support mobile devices. Wang at al [Wang 2014] propose the use of mobile micro-clouds in particular mobility-induced service migration so that computation can follow its mobile users. In a 2014 IEEE article, Sookoor et al propose using smartphones as data collection platforms that leverage mobile high-performance computers deployed on Humvees for data processing [Sookoor 2014]. This work focuses on optimal selection of the processing node based on minimizing job completion time. Even though computation migration and optimal cloudlet selection are two features that are implemented in our tactical cloudlets to promote survivability, the uniqueness of our work is the combination of lightweight versions of these features with additional cloudlet management capabilities that also support very simple cloudlet deployment in the field.

2.2 Software Systems Survivability

Survivability is traditionally defined in military systems as the capability to avoid or withstand the interaction between a system and a given hostile environment [Richards 2007]. More specifically, the Air Force Cyber Vision 2025 document lists survivability as an enduring principle and defines it in terms of fitness/readiness, awareness, anticipation, speed (i.e., responsiveness), agility, and evolvability [Maybury 2013]. While this is only one definition of survivability, it is an example of the many system attributes that fall under the category of survivability.

Survivability of software systems is viewed differently among research and practice communities, but in general, it refers to the capability of a system to continue functioning in spite of adversity. We list some of these definitions below.

- The Self-Healing Systems (Autonomic Computing) community defines survivability as the capability of a system to continue functioning in spite of system disruptions. More specifically, self-healing is the capability to discover, diagnose, and react to system disruptions with

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

the goal of maximizing availability, survivability, maintainability, and reliability of a system [Neti 2007, Salehie 2009].

- The Security community defines survivability as the capability of a system to continue functioning in spite of faults caused by cyber-attacks and to continue to provide essential services, even if in degraded mode [Serageldin 2013, Ellison 1999, Youn 2011, Jarzombeck 2011, Pal 2010, & Park 2006].
- The Software Architecture community views survivability as a system quality that is related to other system qualities such as dependability, availability, reliability, fault-tolerance, and trustworthiness [Avizienis 2013, Sommerville 2004, Xu 2005, Banerjee 2005, & Pokharel 2010].

Our work takes the view of the Software Architecture community, which is to define architectural elements that promote survivability. In particular, this view is aligned with Thuraisingham et al, who simply define survivability as the ability to adapt to changing environments [Thuraisingham 1999].

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

3 Tactical Cloudlets

Forward-deployed, discoverable, virtual-machine-based tactical cloudlets can be hosted on vehicles or other platforms to provide infrastructure to offload computation, provide forward data-staging for a mission, perform data filtering to remove unnecessary data from streams intended for dismounted users, and serve as collection points for data heading for enterprise repositories. The forward-deployed, single-hop proximity to mobile devices promotes energy efficiency as well as lower latency (i.e., faster response times) [Lewis 2014b].

The architecture for our tactical cloudlet implementation is presented in Figure 1. The main elements of the architecture are

- Client: Mobile device running Android 4.x that hosts three main components:
  - Cloudlet-Ready App(s): Mobile apps that are set up to offload computation or data to a cloudlet.
  - Cloudlet Client GUI: Mobile app that is used to access the app store capability described in Section 4.1 of this report.
  - Cloudlet Client Lib: Library that is used by the two components above to discover cloudlets, retrieve data from cloudlets, and offload computation or data. It interacts with the Cloudlet Host using HTTP.
- Cloudlet Host: Linux server that runs a tactical cloudlet. The main components are
  - PyCloud Lib: Python component that implements the core cloudlet functionality.
  - Cloudlet API: Python component that is used by the Cloudlet Client Lib to start Services as Service VMs.
  - Cloudlet Manager: Python web application that is used by an administrator to manage Services (along with their VM Images) and Cloudlet-Ready Apps.
  - Service Repository: Each capability that is made available to mobile apps is considered a Service (see Section 4.1). Each service has associated metadata (i.e., Service Metadata), the actual capabilities packaged as VM disk and memory images (VM Images), and one or more Cloudlet-Ready Apps that can use the capability.
- Admin (PC): Browser that is used to access the Cloudlet Manager web application.

The implementation and additional documentation for tactical cloudlets is available on GitHub at https://github.com/SEI-AMS/pycloud.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

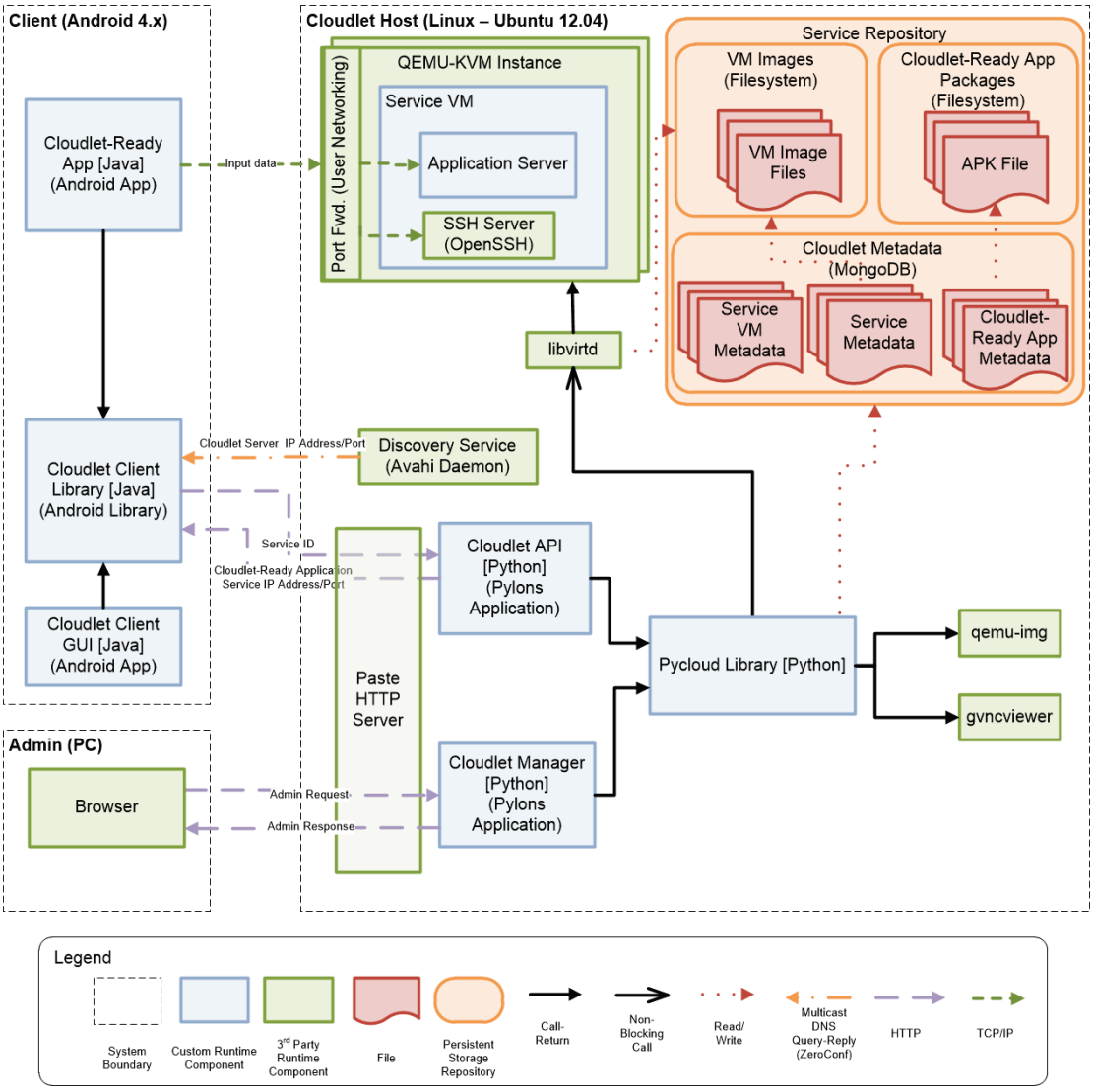

Figure 1: Tactical Cloudlet Architecture

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

4 Tactical Cloudlet Features That Promote Survivability

Sheard states that implementing survivability in a software system requires knowledge of known or predictable threats [Sheard 2008]. Banjeree et al state that survivability deals with three basic kinds of threats: attacks, failures, and accidents [Banerjee 2005]. However, systems in tactical environments must deal with an additional threat—the environment itself. To account for the characteristics of tactical environments, we mapped them to system requirements and then to cloudlet features, as shown in Table 1. Each of these features is described in the following subsections.

Table 1: Mapping of Cloudlet Features to Survivability Requirements

Threats	Intermittent Cloudlet-Enterprise Connectivity	Mobility	Limited Battery Power	Dynamic Missions	Limited Technical Skills in the field
System Requirements	*Disconnected operations*	*Quick response time such that a mobile device receives a response before moving out of range*	*Low energy consumption*	*Ease of re-deployment*	*Ease of deployment*
Tactical Cloudlet Features					
Pre-Provisioned Cloudlets with App Store	X	X	X	X	X
Standard Packaging of Service VMs				X	X
Optimal Cloudlet Selection	X	X			
Cloudlet Management Component				X	X
Cloudlet Handoff/Migration		X		X	

4.1 Pre-Provisioned Cloudlets with App Store

In previous work [Lewis 2014b], we presented several options for cloudlet provisioning and elected to combine pre-provisioning (*Cached VM*) with app store capabilities (Cloudlet Push). This combination leads to lower energy consumption on the mobile device during provisioning, places fewer requirements on mobile devices, and simplifies provisioning in tactical environments. An additional advantage of combining both mechanisms is that if the mobile device already has the client app, it can simply invoke the matching Service VM (if not, it can obtain the client app from the cloudlet, similar to accessing an app store, and then invoke the matching Service VM). The tradeoff is that this combination relies on cloudlets that are pre-provisioned with server capabilities that might be needed for a particular mission or that the cloudlet is connected

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

to the enterprise, even if just at deployment time, to obtain the capabilities. We argue that this requirement is not unreasonable in tactical environments and that it makes cloudlet deployment, and re-deployment, in the field easier and faster while leveraging the state of art and best practices from the cloud computing industry. The following characteristics provide the rationale for meeting system requirements:

- **Capabilities as Services**: Based on the definition of a service in the context of service orientation, each Service VM provides a self-contained capability and exposes a simple interface. Service Metadata is used by the cloudlet discovery protocol to inform mobile devices of available capabilities (see Section 4.3).
- **Virtual Machines as Service Containers**: VMs can be started and stopped as needed based on the number of active users (which is typically bounded in tactical environments because group size is known), therefore providing scalability and elasticity to efficiently support as many active users as possible.
- **Request-Response Nature of Interactions Between Clients 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 and has minimal effect on mobile devices if computation needs to be restarted or migrated.

4.2 Standard Packaging of Service VMs

To support ease of deployment and re-deployment of cloudlets, there is a standard format for Service VMs so they can be easily loaded from the cloudlet disk drive, an enterprise Service VM repository, a thumb drive, or a mobile device connected via USB to the cloudlet. The file format is .csvm, which is a .tar.gz file that contains a JSON file with Service metadata and two image files (disk and state) that compose the VM Image associated to the Service.

- Service Metadata
  - Mandatory
    - Service ID (string): unique identifier for the service; recommended use of a reversed-URL string, similar to the ID used to identify Java packages
    - Service Port (integer): port on which the server, inside the VM, will be listening for connections
  - Optional
    - Version (string): version number of the service
    - Description (string): text description of the service
    - Tags (string): tags that describe the service. These tags are used by the Cloudlet Client Lib service query capability
    - Number of Clients Supported (integer): number of clients the server is designed to support (If more than one, a Cloudlet server may use running Service VMs to serve multiple clients. Otherwise, it will spawn a separate Service VM for each client.)
    - Minimum Memory (MB) (decimal): minimum amount of RAM that the Service requires to run properly
    - Ideal Memory (MB) (decimal): ideal amount of RAM for the Service to run

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

- VM Image: corresponds to two image files—one for the disk image and one for the state/memory image—that contain a suspended Service VM (This VM Image is used to create Service VM Instances.) In particular
  - Disk Image file (.qcow2): qcow2 file containing an image of the hard drive of a Service VM (It contains the installed server files of the program that provides the service.)
  - VM State Image file (.lqs): VM state image in the format that libvirt.save() generates when saving a memory image (The 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 memory state includes the running server listening on the defined port to provide the required service.)

4.3 Optimal Cloudlet Selection

In a cyber-foraging scenario, there may be more than one cloudlet available for use. In this case, it would be useful to know the characteristics of the available cloudlets so that the "best" cloudlet for a Cloudlet-Ready App can be used. To make this selection, we extended our cloudlet discovery protocol so that cloudlet metadata could be used by the Cloudlet Client Lib to execute an algorithm that selects the "best" available cloudlet. One of the goals of the architecture and implementation is to be able to easily change cloudlet metadata as well as the algorithm for cloudlet selection.

The discovery protocol used by our tactical cloudlets uses Zeroconf (www.zeroconf.org), which in turn uses DNS Service Discovery (DNS-SD) along with Multicast DNS. DNS-SD uses DNS queries and replies in a specific format defined for service discovery. A multicast address is used to allow a client to request a specific service without knowing the IP addresses of the available cloudlets (equivalent to a more focused broadcast request).

The Zeroconf discovery protocol is limited in the type and size of the information that can be returned. In particular, broadcast information is static. The data needed by Cloudlet-Ready Apps to evaluate cloudlets, such as CPU, memory, and disk space, is data that must be dynamically collected/calculated by the server and therefore is difficult to broadcast with Zeroconf. To address this difficulty, we added an HTTP GET request for metadata that is invoked after a cloudlet has been discovered. The cloudlet responds with JSON-formatted metadata.

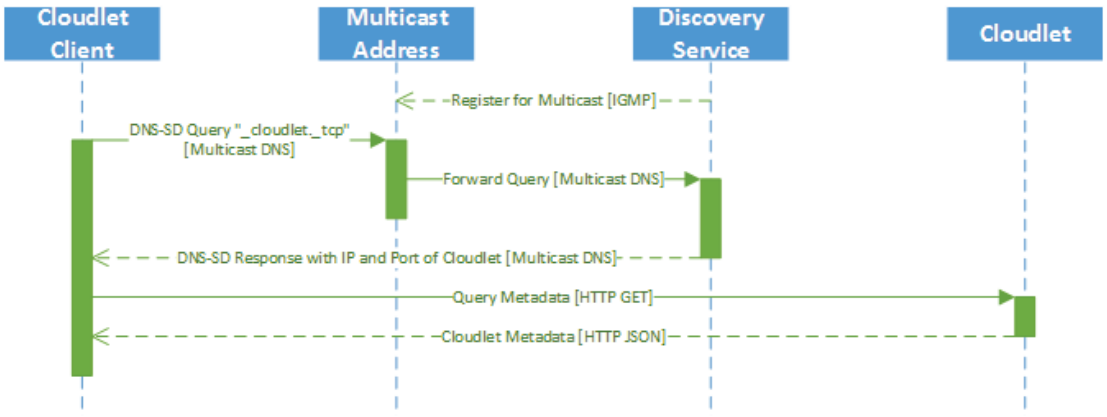

Figure 2: Simplified Sequence Diagram of the Discovery Process

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

The process is defined as

1. When the cloudlet starts, its Discovery Service (the Avahi Daemon in our implementation) joins a particular Multicast IP Address as a listener (Cloudlet Multicast IP). This join is only done when the cloudlet is started regardless of how many discovery queries are received.
2. When the application using the cloudlet client library wants to discover cloudlets, it sends a DNS-SD query for cloudlet services through Multicast DNS to the Cloudlet Multicast IP address.
3. The query reaches the Discovery Service of all cloudlets in the network through Multicast DNS.
4. Each Discovery Service replies with a DNS-SD response indicating the IP and port of its cloudlet.
5. The client requests cloudlet metadata through an HTTP GET request to each cloudlet that it has discovered.
6. Each cloudlet collects the required metadata and replies to the HTTP request with the information.

The cloudlet metadata model consists of a number of objects that are transmitted in JSON format. These data objects are calculated when requested from the client and indicate the current state of the cloudlet, allowing the client to judge if the cloudlet is a good candidate as a destination for offloading its work. The current cloudlet metadata fields are

- CPU metadata (assumes all cores are equal)
  - **cpu.usage** (float, in %): % of CPU in use on the cloudlet, as a whole
  - **cpu.maxCores** (int): number of cores in the system
  - **cpu.speed** (double, MHz): frequency that the cores are working at
  - **cpu.cache** (int, KBs): the amount of cache per core
- Memory metadata
  - **memory.maxMemory** (int, bytes): total amount of physical memory on the cloudlet
  - **memory.freeMemory** (int, bytes): amount of free memory on the cloudlet

The selection of a cloudlet among all the cloudlets discovered on a network is done by the Cloudlet Client Library used by a Cloudlet-Ready App on the mobile device (Figure 1). The cloudlet provides only enough information for the mobile device to use a selection algorithm to select a cloudlet.

Any algorithm implemented by the Cloudlet Client Library has to follow these rules:

- It will receive as input information about the cloudlet (the metadata described above) and information about the service.
- It has to return a double with an evaluation of the cloudlet. This value can be used to compare that cloudlet to others. Higher values mean “better” cloudlets, according to that particular algorithm.

The first ranking algorithm we implemented is called CPUBasedRanker. It evaluates the cloudlet metadata to calculate the percentage of CPU power available to execute the offloaded code. This calculation is done using the formula

$$(100.0 * \text{cpu.maxCores}) - \text{cpu.usage}$$

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

This number represents the percentage of CPU available on the cloudlet at that particular moment.

We performed experiments to evaluate the behavior of the CPUBasedRanker. These experiments were performed on three machines acting as cloudlets; the machines had the following characteristics:

- **Cloudlet 1**
  i7-4700MQ @ 2.40GHz
  32 GB
  0.1% base CPU load
  1.5 GB base memory usage
  Ubuntu 12.04.4 LTS
- **Cloudlet 2**
  Core 2 Q9550 @ 2.83Ghz
  4 GB
  0.2% base CPU load
  1.1 GB base memory usage
  Ubuntu 14.04 LTS
- **Cloudlet 3**
  Core 2 Q9550 @ 2.83Ghz
  32 GB
  0.1% base CPU load
  0.7 GB base memory usage
  Ubuntu 14.04 LTS

The purpose of the experiments was to evaluate the response time of a request to a cloudlet and test if the algorithm correctly selected the less loaded cloudlet for each scenario. We used three test apps: FACE-OPENCV (face recognition), SPEECH (speech recognition) and OBJECT (object recognition). The results of the experiment are shown in Table 2. App-Ready Time measures the roundtrip request-response time in seconds of the first request, including the setup of the Service VM on the cloudlet. Average Response Time measures the average roundtrip request-response time in milliseconds of each subsequent request (all the Service VMs are capable of handling multiple users). Each row corresponds to a different scenario in which the three cloudlets in the network are working under different loads.

Table 2 shows that the CPUBasedRanker indeed always selects the cloudlet with the lowest load. However, it does not always select the cloudlet that would deliver the smallest response time because not all the cloudlets in our experiments were equal. Cloudlet 1 is the most powerful cloudlet and has the smallest app-ready and response times. Data from additional experiments showed that even at higher loads Cloudlet 1 would still have smaller app-ready and response times than Cloudlets 2 and 3. Therefore, the CPUBasedRanker is ideal only if all cloudlets have the same characteristics.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Table 2: Cloudlet Selection and Execution Data

CPU Load (%)			FACE-OPENCV			SPEECH			OBJECT		
Cloudlet 1	**Cloudlet 2**	**Cloudlet 3**	**Selected Cloudlet**	**App-Ready Time(s)**	**Avg. Resp. Time (ms)**	**Selected Cloudlet**	**App-Ready Time(s)**	**Avg. Resp. Time (ms)**	**Selected Cloudlet**	**App-Ready Time(s)**	**Avg. Resp. Time (ms)**
0	50	95	1	3.41	11	1	3.37	3,083	1	3.07	1,815
50	0	95	2	15.65	9	2	20.09	5,374	2	21.21	3,649
50	95	0	3	18.13	18	3	19.73	4,575	3	16.00	3.970
0	95	50	1	2.67	10	1	2.65	3.054	1	3.13	2.066
95	0	50	2	14.99	10	2	18.96	4,049	2	20.81	3,844
95	50	0	3	24.84	9	3	19.17	5,017	3	15.70	4,170

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

This experiment led us to another approach for a ranking algorithm, which is to calculate a value that measures the overall performance of the cloudlet. There are different ways to define and measure this value:

- **Benchmarking**: The most widely used way of measuring the performance of a computer is by using a benchmark. A benchmark is a program that can be run on a computer that calculates a number that defines how well the device performs a given set of instructions. This number, while not meaningful by itself, can be used to compare the performance between different computers. There are many benchmarks; some focus on the CPU, some on memory, others on both, and still others on more specific aspects. If a benchmark program is used to measure the performance of a cloudlet, the performance value would then be stored on the cloudlet and could be sent to the mobile device for cloudlet selection. The advantages of this method are that (1) it is an established method for measuring computer performance and (2) there are many available tools. Disadvantages are that it requires a tool to be installed on the cloudlet, adding more dependencies. The benchmark tool must be executed on installation, making the installation process a bit more complex. Also, the benchmark would not account for the specifics of the service being requested, which may need more CPU than memory, or only memory, etc.
- **Profiling**: Profiling can be used to measure how well a service executes on a given cloudlet. This way of measuring value could simply measure how fast a service is able to complete certain requests (using enough runs and calculating averages). This value must be stored on the cloudlet after the profiling process. It can be sent to a mobile device for selecting a cloudlet that best matches the service that it is requesting. The main advantage of this method is that it takes into account the characteristics of the service being requested. The main disadvantage is that if the input for a service varies considerably, unless the profiling is very exhaustive, the value may only be appropriate for certain inputs. And this method requires profiling each service installed on the cloudlet after installing it, making service installation more complex, which could potentially take a considerable amount of time.
- **Formula**: Different characteristics of the cloudlet can be used in a formula that calculates a value that tries to approximate the overall performance of the cloudlet. The main advantage of this approach is that it does not require additional processes or tools on the cloudlet side, other than gathering and sending the information as cloudlet metadata to the mobile device. However, there is no standard formula that approximates the performance of a computer based on some of its characteristics; however, an approximation could be obtained with several experiments, measuring response time as the target value to minimize. Unless different formulas are created depending on the service, this approximation does not account for the specifics of the service being requested, which may need more CPU than memory, or only memory, etc.

Another option is not to calculate a single overall performance value, but to calculate a value for CPU power or memory only to measure the computational performance or the memory performance of the system, respectively. Because a Cloudlet-Ready App can choose the selection algorithm to use, it could select based on the feature that is more useful for its particular needs. As such, we could have the following ranking algorithms:

- **CPUPerformanceRanker**: This algorithm could be derived with an over-simplification by measuring performance using the CPU frequency as a proxy. Though this approach is not

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

very accurate, it could be used as a rough comparison. A simple formula that could be used to calculate the *available CPU performance* of a cloudlet is

((100.0 * cpu.maxCores) - cpu.usage) * cpu.speed

This formula is the same one used to calculate the percentage of available CPU performance in the CPUBasedRanker, but uses the percentage to calculate the "available speed" for the offloaded code.

- **MemoryPerformanceRanker**: Information on free memory and CPU cache could be combined to measure the *available memory performance* of a cloudlet. The formula could be

  (cpu.cache * cpu.maxCores) * A + memory.freeMemory * B

  where A and B are constants that could be calculated to give different weights to cache and RAM.

We are currently in the process of implementing and testing these two ranking algorithms.

It could be possible to calculate a combined model, based on the two algorithms above, to obtain *overall performance*. A linear formula for this could be

CPUPerformanceRanker * C + MemoryPerformanceRanker * D

where C and D are constants. Experiments would have to be performed to find appropriate values for these constants that minimize response times. However, because the value would be the same for any service and not account for the requirements of the requested service, it could end up being less useful than having separate ranking algorithms.

4.4 Cloudlet Management Component

The Cloudlet Manager shown in Figure 1 is a lightweight, web-based interface that enables easy deployment and redeployment of capabilities. It provides the following functionality:

- Service VM creation, edit, and deletion
- Service VM import and export
- Service VM Instance start, stop, and migration
- Cloudlet-Ready App repository (i.e., app store)

Figure 3 is a screenshot of the Cloudlet Manager GUI that shows four available services on the cloudlet, their Service ID, and the port they are listening on. The buttons to the right of each service are used to start a Service VM Instance of the service, edit a service, remove a service, and export a service. In essence, the Cloudlet Manager provides console capabilities similar to that of cloud providers but at a much smaller scale.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

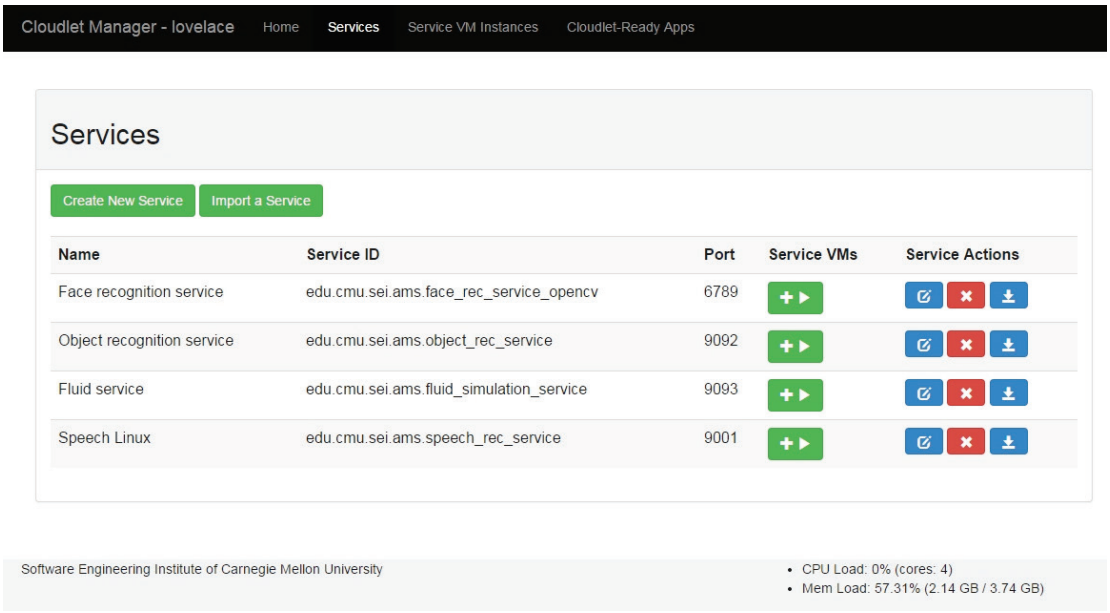

Figure 3: Cloudlet Manager Screenshot

4.5 Cloudlet Handoff/Migration

A characteristic of cloudlets in resource-constrained environments is that (1) they can be mobile because they can reside on vehicles and (2) clients can easily become disconnected due to mobility and intermittent network connectivity. A strategy to deal with these problems is to enable manual handoff of data and computation between cloudlets that are within range of each other. Manual handoff would enable scenarios in which a user can migrate capabilities from a fixed cloudlet to a mobile cloudlet to support field operations as well as reintegrate capabilities back to the fixed cloudlet.

Our tactical cloudlet implementation uses QEMU+KVM through libvirtd, as shown in Figure 1. The migration process leverages libvirt to control the migration of a Service VM, as shown in Figure 4. The steps of the migration process are

1. The Cloudlet Server (pycloud) sends Service VM Instance Metadata to the Remote Cloudlet Server.
2. The Remote Cloudlet Server (pycloud) adds the Service VM Instance Metadata to its list of Service VM Instances in the Service Repository.
3. The Cloudlet Server pauses (suspends) the Service VM Instance so that the disk image is not changed while it is being sent.
4. The Cloudlet Server sends the Service VM Instance disk image file to the Remote Cloudlet Server.
5. The Remote Cloudlet Server points (rebases) the Service VM Instance Disk Image File to the path of the Service base VM image file on that Remote Cloudlet.
6. The Cloudlet Server calls the libvirtd function to migrate the VM (XML description and state of the VM, and memory).
7. The Remote Libvirt Daemon receives and stores the VM.
8. The Cloudlet Server notifies the Remote Cloudlet Server that the migration has finished.
9. The Remote Cloudlet Server resumes the migrated Service VM Instance.

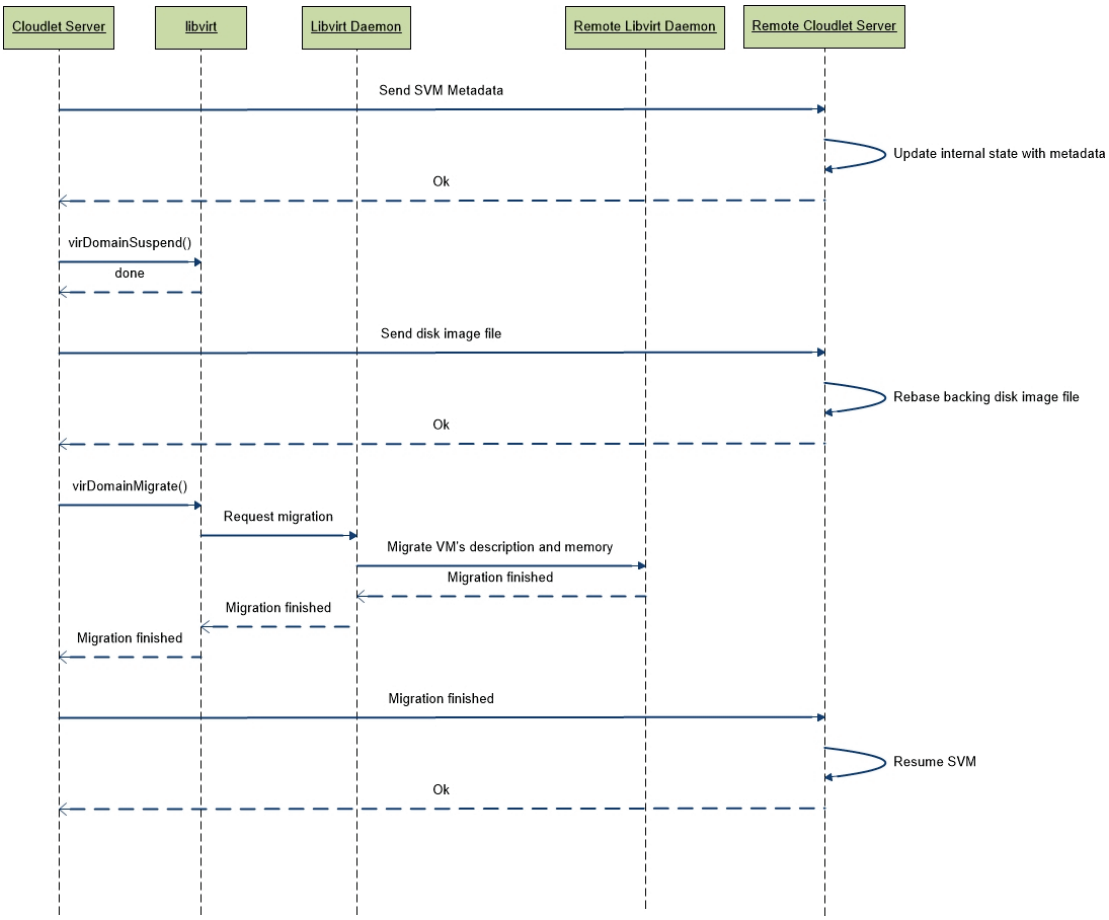

Figure 4: Service VM Migration Process

To test migration times, we used two of the test apps from Section 4.3—SPEECH and OBJECT—plus a third app called FLUID, which is a fluid simulation app. The reason for including FLUID is because it is a continuous interaction app rather than a request-response app, so we could see the effects of migration on this type of application. Migration was performed between Cloudlet 1 and Cloudlet 3 as defined in Section 4.3.

Both cloudlets were connected to the same switch with a 1Gbps link. The roundtrip time for a ping between the two machines is on average 0.2 ms. Service VM Instance image file sizes and average migration times for the test applications are shown in Table 3. The VM disk image is not a full image, but a qcow2 Service VM instance image that contains only the differences between the disk image of the Service and the current running instance. Migration time is measured from the moment the migration button is pushed in the Cloudlet Manager until the Service VM is running again on the Remote Cloudlet Server.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Table 3: VM Migration Data

App	Service VM Instance Disk Image Size (MB)	Service VM Instance Memory Image Size (MB)	Average Migration Time (s)
SPEECH	8.6	492	6.33
OBJECT	8.7	442	6.23
FLUID	13	919	5.47

The VM image file sizes and VM migration times are approximately the same. For the SPEECH and OBJECT apps that have a request-response interaction with the cloudlet, the user simply experiences a longer response time if executing the application at the moment of migration. For the FLUID application, which constantly receives data from the cloudlet (continuous interaction), the app freezes momentarily during the migration. The migration times for FLUID are slightly better than for the other cases, even though FLUID has the largest VM image sizes. This result may be caused by the way in which the memory state is transferred, which is done directly by the libvirt daemons without creating a saved state file. The information in the FLUID memory state may be easier to compress or to efficiently transfer than for the other apps, thus decreasing the migration times slightly.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

5 Conclusions and Next Steps

This report presents an architecture and experiment results that demonstrate that cyber-foraging using tactical cloudlets increases the survivability of mobile systems, which we define as the capability of a system to continue functioning in spite of adversity. We mapped the characteristics of tactical environments to system requirements for survivability and then to tactical cloudlet features.

The next steps in this area of our research are to (1) develop and evaluate a set of rankers for different service characteristics, and (2) add capabilities for automated migration that would enable load balancing, similar to what is done in cloud data centers for resource optimization, or enable migration to a more powerful or nearby cloudlet to improve response time and provide continued operations. In other areas of our research, we are exploring trusted identities for secure communications and the use of tactical cloudlets for efficient data staging.

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Bibliography

[Avizienis 2013]
Avizienis, Algirdas, et al. Basic Concepts and Taxonomy of Dependable and Secure Computing. *Dependable and Secure Computing, IEEE Transactions*. Volume 1. Number 1. October 2013. Page 11. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1335465

[Banerjee 2005]
Banerjee, Somo, et al. Leveraging Architectural Models to Inject Trust into Software Systems. Pages 1-7. In *Proceedings of the 2005 Workshop on Software Engineering for Secure Systems—Building Trustworthy Applications (SESS 2005). ACM.* St. Louis, Missouri. May 2005. http://dl.acm.org/ft_gateway.cfm?id=1083213&ftid=328196&dwn=1&CFID=573835016&CFTOKEN=50705130

[Ellison 1999]
Ellison, Robert J., et al. Survivable Network System Analysis: A Case Study. *IEEE Software*. Volume 16. Number 4. July 1999. Page 70. http://www.computer.org/csdl/mags/so/1999/04/s4070.pdf

[Jarzombeck 2011]
Jarzombeck, Joe. Software Assurance: Mitigating Risk of Zero-Day Attacks and Enabling Resilience of Cyber Infrastructure. Pages 5:1-5:1. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW). ACM.* Oak Ridge, Tennessee. October 2011. http://scap.nist.gov/events/2011/itsac/presentations/day1/Jarzombek,%20Millar,%20Banghart%20-%20Mitigating%20the%20Risk%20of%20Zero-Day%20Attacks%20with%20Software%20Security%20Automation.pdf

[Lewis 2014a]
Lewis, Grace, et al. Architecture Strategies for Cyber-Foraging: Preliminary Results from a Systematic Literature Review. *Software Architecture*. Avgeriou, Paris & Zdun, Uwe. [editors]. Springer International Publishing. Pages 154-169. 2014.

[Lewis 2014b]
Lewis, Grace, et al. Tactical cloudlets: Moving Cloud Computing to the Edge. Pages 1440-1446. In *Military Communications Conference (MILCOM). IEEE*. Baltimore, Maryland. October 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6956959

[Maybury 2013]
Maybury, Mark T. *Cyber Vision 2025*. AF/ST TR 12-01. U.S. Air Force. 2013. http://www.defenseinnovationmarketplace.mil/resources/AF_Cyber_Vision_2025.pdf

[Neti 2007]
Neti, Sangeeta & Muller, Hausi. Quality Criteria and an Analysis Framework for Self-Healing Systems. Pages 6-6. In *Software Engineering for Adaptive and Self-Managing Systems, 2007. ICSE Workshops SEAMS'07*. Shanghai, China. May 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&asrnumber=4228606

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

[Pal 2010]
Pal, Partha, et al. Advanced Protected Services—A Concept Paper on Survivable Service-Oriented Systems. Pages 158-165. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW). IEEE.* Carmona, Spain. May 2010. http://ieeexplore.ieee.org /stamp/stamp.jsp?tp=&arnumber=5479512

[Park 2006]
Pal, Joon S., et al. Component Integrity Check and Recovery Against Malicious Codes. Pages 5-pp. In *Advanced Information Networking and Applications, 2006 (AINA 2006). IEEE.* Vienna, Austria. April 2006. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1620423

[Pokharel 2010]
Pokharel, Manish, et al. Disaster Recovery for System Architecture Using Cloud Computing. Pages 304-307. In *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium. IEEE.* Seoul, South Korea. July 2010. http://ieeexplore.ieee.org /stamp/stamp.jsp?tp=&arnumber=5598055

[Richards 2007]
Richards, M.; Hastings, D.; Ross, A. & Rhodes, D. Design Principles for Survivable System Architecture. Pages 1-9. In *Systems Conference, 2007 1st Annual IEEE.* 2007. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4258850&tag=1

[Salehie 2009]
Salehie, Mazeiar & Tahvildari, Ladan. Self-Adaptive Software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*. Volume 4. Number 2. May 2009. Page 14:1. http://dl.acm.org/ft_gateway.cfm?id=1516538&ftid= 622359&dwn=1&CFID=573835016&CFTOKEN=50705130

[Satyanarayanan 2001]
Satyanarayanan, Mahadev. Pervasive Computing: Vison and Challenges. *IEEE*. Volume 8. Number 4. August 2001. Page 10. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=943998

[Satyanarayanan 2009]
Satyanarayanan, Mahadev, et al. The Case for VM-Based Cloudlets in Mobile Computing. *Pervasive Computing. IEEE*. Volume 8. Number 4. October 2009. Page 14. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5280678

[Satyanarayanan 2013]
Satyanarayanan, Mahadev, et al. The Role of Cloudlets in Hostile Environments. *Pervasive Computing. IEEE*. Volume 12. Number 4. October 2013. Page 40. http://ieeexplore.ieee.org /stamp/stamp.jsp?tp=&arnumber=6648598

[Serageldin 2013]
Serageldin, Ahmed, et al. A Survivable Critical Infrastructure Control Application. Pages 34:1-34:4. In *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop. ACM*. Oak Ridge, Tennessee. January 2013. http://dl.acm.org /ft_gateway.cfm?id=2460015&ftid=1361000&dwn=1&CFID=573835016&CFTOKEN= 50705130

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

[Sheard 2008]
Sheard, Sarah. 11.2. 2 a Framework for System Resilience Discussions. *INCOSE International Symposium*. Volume 18. Number 1. June 2008. Page 1243. http://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2008.tb00875.x/pdf

[Sommerville 2004]
Sommerville, Ian. *Software Engineering (7th Edition)*. Pearson Addison Wesley. 2004. ISBN 9780321210265. http://www.pearsonhighered.com/educator/academic/product/0,1144,0321210263,00.html

[Sookoor 2014]
Sookoor, Tamim, et al. Offload Destination Selection to Enable Distributed Computation on Battlefields. Pages 841-848. In *Military Communications Conference (MILCOM). IEEE*. Baltimore, Maryland. October 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6956867

[Thuraisingham 1999]
Thuraisingham, Bhavani M. & Maurer, John. Information Survivability for Evolvable and Adaptable Real-Time Command and Control Systems. *Knowledge and Data Engineering, IEEE Transactions*. Volume 11. Number 1. January 1999. Page 228. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=755631

[Wang 2014]
Wang, Shiqiang, et al. Tactical Cloudlets: Mobility-Induced Service Migration in Mobile Microclouds. Pages 835-840. In *Military Communications Conference (MILCOM). IEEE*. Baltimore, Maryland. October 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6956866

[Xu 2005]
Xu, Lihua, et al. An Architectural Pattern for Non-Functional Dependability Requirements. Pages 1-6. In *Proceedings of the 2005 Workshop on Architecting Dependable Systems (WADS 2005). ACM.* St. Louis, Missouri. May 2005. http://dl.acm.org/ft_gateway.cfm?id=1083219&ftid=328201&dwn=1&CFID=573835016&CFTOKEN=50705130

[Youn 2011]
Youn, Hyunsang, et al. Security Based Survivability Risk Analysis with Extended HQPN. Pages 25:1-25:10. In *Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication (ICUIMC). ACM.* Seoul, South Korea. February 2011. http://dl.acm.org/ft_gateway.cfm?id=1968644&ftid=948478&dwn=1&CFID=573835016&CFTOKEN=50705130

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	February 2016	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Cyber-Foraging for Improving Survivability of Mobile Systems	FA8721-05-C-0003

6. AUTHOR(S)

Sebastián Echeverría (Universidad de los Andes), Grace A. Lewis, James Root, & Ben Bradshaw

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2016-TR-001

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFLCMC/PZE/Hanscom Enterprise Acquisition Division 20 Schilling Circle Building 1305 Hanscom AFB, MA 01731-2116	n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

Cyber-foraging is a technique for dynamically augmenting the computing power of resource-limited mobile devices by opportunistically exploiting nearby fixed computing infrastructure. Cloudlet-based cyber-foraging relies on discoverable, generic, forward-deployed servers located in single-hop proximity of mobile devices. We define tactical cloudlets as the infrastructure to support computation offload and data staging at the tactical edge. However, the characteristics of tactical environments—such as dynamic context, limited computing resources, disconnected-intermittent-limited (DIL) network connectivity, and high levels of stress—pose a challenge for the continued operations of mobile systems that leverage cloudlets in tactical environments. We also define survivability of mobile systems as the capability of a system to continue functioning in spite of adversity. This report presents an architecture and experimental results that demonstrate that cyber-foraging using tactical cloudlets increases the survivability of mobile systems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Cyber-foraging, tactical cloudlets, mobile systems	28

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL