ParaDrop: A Multi-tenant Platform to Dynamically Install Third Party Services On Wireless Gateways

Dale Willis
Department of Computer Science
University of Wisconsin
dale@cs.wisc.edu

Arkodeb Dasgupta
Department of Electrical Engineering
University of Wisconsin
arko@engr.wisc.edu

Suman Banerjee
Department of Computer Science
University of Wisconsin
suman@cs.wisc.edu

ABSTRACT

The landscape of computing capabilities within the home has seen a recent shift from persistent desktops to mobile platforms, which has led to the use of the cloud as the primary computing platform implemented by developers today. However, a growing number of high quality services restrict computational tasks to be colocated with the end-user. Thus, we introduce a specific edge computing framework, called ParaDrop, which allows developers to leverage one of the last bastions of persistent computing resources in the end customer premises: the gateway (e.g., the WiFi Access Point or home set-top box). Using our platform, which has been fully implemented on real hardware, developers can design virtually isolated compute containers to provide an persistent computational presence in the proximity of the end-user. The compute containers, retain user state, and also move with the users as the latter changes their points of attachment. We demonstrate the capabilities of our platform by implementing two third-party applications, which utilize the framework we have provided. The framework we have implemented for the developer allows multitenancy through virtualization, dynamic installation through our developer API, and tight resource control through a managed policy design.

1. INTRODUCTION

Cloud computing platforms, such as Amazon EC2 and Google App Engine, are popular for many reasons including their reliable, always on, and robust nature. The capabilities that centralized computing platforms provide are inherent to their implementation, and unmatched by previous platforms (e.g., Desktop applications). Thus, third-party developers have come to rely on cloud computing platforms to provide high quality services to their end-users.

While the popularity of cloud services has pushed computational tasks further from the end-user, a subset of services puts unyielding constraints on the developer. Over the years, a number of research threads have proposed that a better end-user experience is possible if the computation is performed close to the end-user. This is typically referred to as "edge computing", and comes in various flavors including: Cyber Foraging[1], Cloudlets[6], and more recently Fog Computing[2].

In this paper, we detail our design of an edge computing architecture, which enables third-party developers to deploy their services into the end-user's premises. Our platform, called ParaDrop, allows shared, but controlled, access to resources in end-user gateways in a multi-tenant fashion through the use of a low-overhead virtualization framework.

1.1 Multi-tenant wireless gateways, applications, and our approach

A decade or two ago, the desktop computer was the only reliable computing platform within the home where third-party applications could reliably and persistently run. However diverse mobile devices, such as smart phones and tablets, have deprecated the desktop computer since, and today persistent third-party applications are often run in remote cloud-based servers. While cloud-based third-party services have many advantages, the rise of edge computing concepts stems from the observation that many services can benefit from a persistent computing platform, right in the end-user premises.

With end-user devices going mobile there is one remaining device, which provides all the capabilities developers require for their services, as well as the proximity expected from an edge computational framework. The gateway — which could be a home WiFi Access Point or a cable set-top box provided by a network operator — is a platform that is continuously on, and due to its pervasiveness is a primary entry point into the end-user premises for such third-party services. To keep our implementation lightweight, especially given the limited resources of typical gateways, we use the Linux Container (LXC) abstraction for resource isolation among third-party services. To allow effective management of the networking functions across these gateways and their diverse third-party services, we use an OpenFlow-based framework.

Applications: We consider a number of applications of our ParaDrop framework. The following are a few examples: (i) A security camera vendor needs to collect all video feeds from the different cameras installed in a single home, and implements a motion detection service across multiple rooms that triggers alert messages and a real-time video feed to the home owners. The wireless gateway can be a natural host for such proprietary third-party computation, storage of sensitive personal content, all with a low-latency reach of the installed cameras. (ii) A set of temperature and hu-

MobiArch'14, September 11, 2014, Maui, Hawaii, USA.

http://dx.doi.org/10.1145/2645892.2645901 .

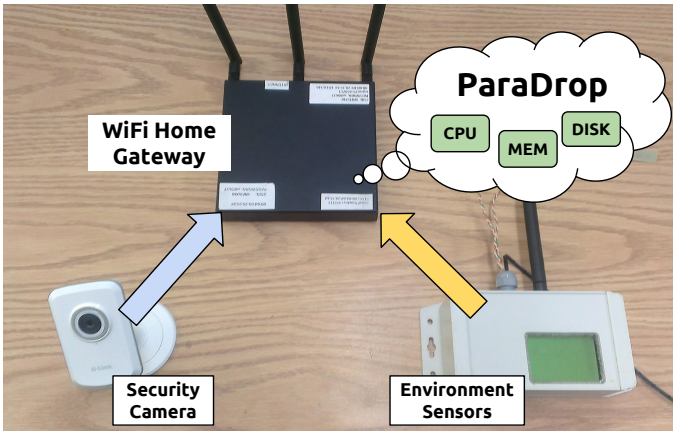

Figure 1: The fully implemented ParaDrop platform on the WiFi Home Gateway, which shares its resources with two wireless devices including a Security Camera and Environment Sensor.

midity sensors are placed in the home that need to interact with the home thermostat to optimally manage the latter's operation. The proprietary logic and the home-specific data can again use ParaDrop to be suitably installed in the home gateway. (iii) A streaming media service, e.g., Netflix, can install their proprietary software in the gateways, through which they can selectively download and cache the top five recommended movies of the home users, within the gateway. By doing so ahead of the viewing time, Netflix can lessen their real-time bandwidth demands during peak periods. In each of these examples, the third-party developers (the security camera company, the temperature and humidity sensing company, or Netflix) would be allowed to dynamically install their proprietary code *on-demand* into the user-specific Linux containers (LXCs) in each gateway. *As users move from one location to another, the user-specific LXCs can move with them to their nearest home gateway, thus maintaining close proximity of the local state.*

1.2 A developer-centric framework

In this paper we examine the requirements of services in order to build an edge computing platform, which enables developers to provide services to the end-user in place of a cloud computing platform. A focus on edge computation would require developers to think differently about their application development process, however we believe there are many benefits to a distributed platform such as ParaDrop.

The developer has remained our focus in the design and implementation of our platform. Thus, we have implemented ParaDrop to include a fully featured API for development, with a focus on a centrally managed framework. Through virtualization, ParaDrop enables each developer access to resources in a way as to completely isolate all services on the gateway. A tightly controlled resource policy has been developed, which allows fair performance between all services.

1.3 ParaDrop Capabilities

ParaDrop takes advantage of the fact that resources of the gateway are underutilized most of the time. Thus each service, referred to as a *chute* (as in parachute), borrows CPU time, unused memory, and extra disk space from the gateway. This allows vendors an unexplored opportunity to provide added value to their services through the close proximity footprint of the gateway.

Figure 1 shows our system running on real hardware, the "WiFi Home Gateway", along with two services to motivate our platform: "Security Camera" and "Environment Sensors". ParaDrop has been implemented on a PCEngines ALIX 2d2 Single Board Computer running OpenWrt "Barrier Breaker" on an AMD Geode 500MHz processor with 256MB of RAM. This low-end hardware platform was chosen to showcase ParaDrop's capabilities with existing gateway hardware.

We have emulated two third-party developers who have migrated their services to the ParaDrop platform to showcase the potential of our system. Each of these services contain a fully implemented set of applications to capture, process, store, and visualize the data from their wireless sensors within a virtually isolated environment. The first service is a wireless environmental sensor designed as part of the Emonix research platform[4], which we refer to as "EnvSense". The second service is a wireless security camera based on a commercially available D-Link 931L webcam, which we call "SecCams".

Leveraging our platform, the two developer services allow us to motivate the following characteristics of ParaDrop:

i) Privacy: Many sensors and even webcams today rely on the cloud as the only storage mechanism for generated data. Leveraging the ParaDrop platform, the end-user no longer must rely on cloud storage for the data generated by their private devices, and instead can borrow disk space available in the gateway for such data.

ii) Low Latency: Many simple processing tasks required by sensors are performed in the cloud today. By moving these simple processing tasks onto gateway hardware, one hop away from the sensor itself, a reliable low latency service can be implemented by the developer.

iii) Proprietary Friendly: From a developer's perspective, the cloud is the best option to deploy their proprietary software because it is under their complete control. Using ParaDrop, a developer can package up the same software binaries and deploy them within the gateway to execute in a virtualized environment, which is still under their complete control.

iv) Local Networking Context: In the typical service implemented by a developer, the data is consumed only by the end-user, yet stored in the cloud. This requires data generated by a security camera in the home to travel out to a server somewhere in the Internet, and upon the end-user's request travel back from this server into the end user device for viewing. Utilizing the ParaDrop platform, a developer can ensure that *only* data requested by the end user is transmitted through Internet paths to the end user device.

v) Internet Disconnectivity: Finally, as services become more heterogeneous they will move away from simple "nice to have" features, into mission critical, life saving services. While generally accepted as unlikely, a disconnection from the Internet makes a cloud based sensor completely useless, and is unacceptable for services such as health monitoring. In this case, a developer could leverage the always-on nature of the gateway to process data from these sensors, even when the Internet seems to be down.

Contributions: By providing third-party developers to leverage ParaDrop to deploy flexible services, we make the following contributions:

- We define a specific framework which enables third-party software services to be installed directly on the gateway;
- We provide a complete implementation of ParaDrop on existing WiFi home gateways, including the use of lightweight virtualization through Linux containers and SDN-based network management, and deploy two fully featured third-party applications: a video security service running on a commercially purchased webcam, and a home environment sensing service running on a sensor network research platform;
- We define an API for third-party developers to distribute their services via our platform, which will be open sourced and publicly available; and
- We present initial results from our deployment, which show the efficiency and fairness that ParaDrop provides through tight resource policy controls on the gateway.

2. THE PARADROP PLATFORM

While implementing the platform, we focused on transparent capabilities when possible, which are stated through the following platform goals:

1) ParaDrop should not effect the networking performance of the *host* system.
2) Full isolation should be maintained between each chute, including performance.
3) Chutes can be dynamically manipulated, thus allowing on-the-fly migration and optimization of services.

We now focus our attention on the platform characteristics of ParaDrop including how services are implemented by the developer, managed by an administrator, and viewed by the end-user. In order to describe these characteristics for sections 2.1-2.3 below, we will refer to Figure 2.

2.1 A Developer's View

We motivated ParaDrop by implementing two unique services for the end-user, both of which provide characteristics of both a cloud-based and edge-computational device. For each service the developer's requirements will be unique, and therefore we detail the services below.

2.1.1 The EnvSense Service

The wireless sensor for EnvSense is based on an existing research platform currently deployed in buildings, which monitors temperature and humidity data. Since the service is fully implemented, we had the opportunity to *migrate* the applications, rather than rewrite them to fit within our framework. From Figure 2, we can see that the developer is required to provide two capabilities: configuration and web services.

Configuration of the chute enables the developer to specify details about how the virtualized environment should be structured. This includes how it will communicate through networking interfaces to both its sensors and the Internet, how much storage it requires, as well as any custom applications written by the developer.

Using the ParaDrop framework, the developer was able to migrate all applications to execute in their chute. The applications, typically running on a server for the sensors, provides data storage, timestamp synchronization, and data visualization capabilities for the environmental sensor. Without changing a single line of code (including C, Perl, Python, and PHP) we were able to successfully migrate these services to execute within a chute.

2.1.2 The SecCams Service

The SecCams service is based on a commercially available wireless IP camera, where we took the role of developer to fully implement the service. This allowed us the experience of building a service from scratch, while thinking about the distributed control resulting from the use of the ParaDrop platform.

Configuration of the SecCams chute was very similar to the EnvSense chute. We required networking interfaces to communicate with the webcam and the Internet, as well as ample storage for images. The storage was possible because of a flash card we added to our gateway device which provides GBs of storage.

The applications for SecCams allow for motion detection from the webcam, user defined alerts, as well as visualization of the detected images. The motion detection was a Python based program with user defined characteristics such as threshold of motion, time of day, and rate of detection. Visualization of the motion was implemented as a PHP based web page, which was hosted within the chute.

2.2 An Administrator's View

In order to maintain our platform goals of both performance isolation between chutes, as well as transparency to the host network, a strict resource policy must be enforced by ParaDrop. We have implemented ParaDrop with the ability to have a network operator (or ISP) manage the gateway resources allocated to each chute. The network operator is the perfect management entity of ParaDrop for two reasons. First of all, end-users must go through an ISP for access to the Internet, as well as typically providing the networking equipment on which ParaDrop is implemented. Second, the ParaDrop service must be provided to developers, since the ISP is a middleman between developers and end-users, they are the natural choice.

Figure 2 shows the network operator communicating with the `Management Interface`, which is populated by the ParaDrop server. This interface details the current resource usage of the gateway, as well as the percentage of resources each chute is utilizing. If a situation comes up where one chute is unfairly utilizing a particular resource (e.g., CPU time), or effecting the host network throughput, the network operator has the ability to adjust the resource policy on demand.

2.3 A User's View

Services should be transparently presented to the user, as if they were cloud-based. Therefore, when the user adds a new service, they register with the developer as they normally would. Likewise, when they want to view their service's data, they would log on to the developer's web page.

Depending on the location of the user (e.g., connected to LAN of gateway or outside the house), and how the developer has implemented their services, will determine how the

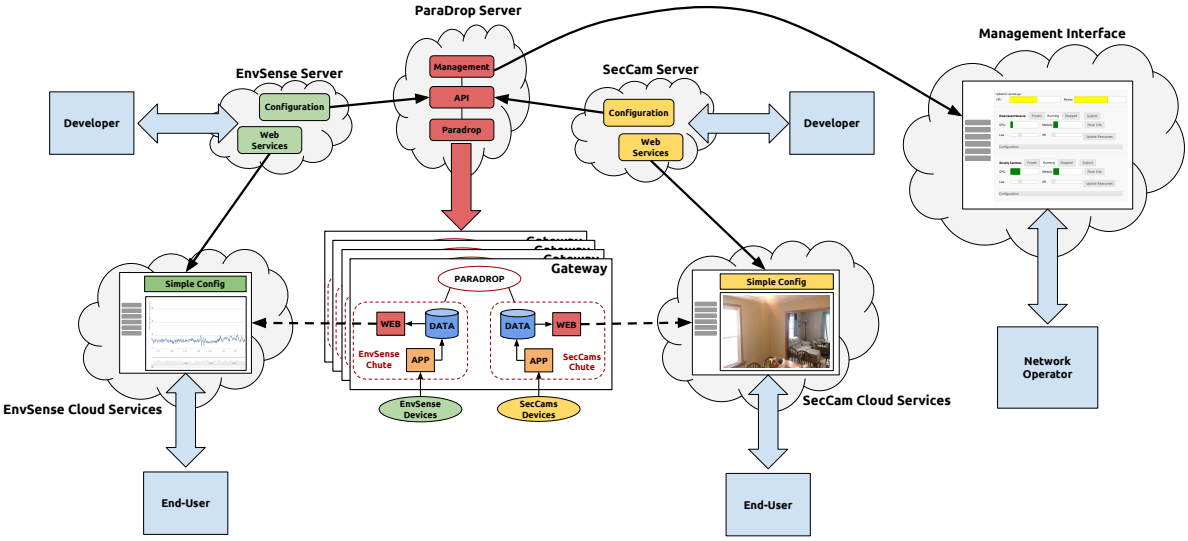

Figure 2: The full ParaDrop platform. Many gateways communicate with the centralized ParaDrop servers, and all developers and network operators communicate with their devices through the ParaDrop API.

data is presented to the end-user. Using the ParaDrop platform, the developer has the choice to decouple the web page and the data from each other. What this allows is a web page, which can be hosted from the cloud or within the gateway, as well as the sensor data coming from the gateway or via a proxy through the cloud.

3. IMPLEMENTATION ASPECTS

There are several primary concerns of the ParaDrop platform including installation procedure, API, and networking configuration.

Dynamic Installation: In order to allow end-users to easily add services to their gateway, each service should have the ability to be dynamically installed. This process is possible through the virtualization environment of each chute. When an end-user wishes to add a service to their home, they simply register an account with the developer. Using the ParaDrop API, the developer links the user's account with their gateway. If the service utilizes a wireless device, the gateway can fully integrate with the device without any interference from the end-user.

ParaDrop API: The focus of ParaDrop is to enable third-party developers to provide high quality services to their users. In order to enable this, a seamless API was developed, based on a RESTful paradigm, which allows the developer to have complete control over the configuration of their chutes. As services evolve, the API will provide all the capabilities required without the need for modification to the configuration software. This is possible through the use of a JSON based data backend which allows abstract configuration and control over each chute.

OpenFlow: The networking topology of a dynamic, virtualized environment controlled by several entities is very complex. In order to maintain control over the networking aspects of the gateway, we leverated an SDN paradigm through the use of OpenFlow with a Floodlight controller backend. All configuration related to networking between the chutes and the gateway are handled through a Floodlight module, which is interfaced by the developers and network operators. The use of SDN is what allows developers to transparently redirect the user's request to their web services from within the gateway.

Resource Policy: The multitenancy aspects of ParaDrop require tight policy control over the gateway and its limited resources. Currently the major resources controlled by ParaDrop include: CPU, memory, and networking. Using the API, the developer specifies the type of resources they require depending on the services they implement. Through the management interface, the network operator, can dynamically adjust the resources provided to each chute. These resources are adjusted first by a request sent to the chute, and if not acted upon, then by force through the virtualization framework tools.

Test	Host	LXC	Lguest
Start Time (sec)	-	2	40
Packet Latency (ms)	0.04	0.20	39.0
Throughput Fairness	Host	50%	60%/40%
	Chute	50%	30%/30%

Table 1: Overheads of virtualization methods

4. SYSTEM EVALUATION

There are a few main characteristics of ParaDrop, which were evaluated during the design phase of our platform.

4.1 Virtualization

As the backbone to our multitenant approach, the capabilities provided by virtualization were very important. We implemented two different virtualization methods, before settling on a particular one, which provides lower overhead at the cost of a few specific features.

In the first iteration of our design, we implemented "Lguest", a paravirtualization hypervisor supported in the mainline Linux kernel. This is a simple hypervisor, and in fact the only one supported by our AMD Geode processor, due to

its older feature set. Lguest provided many capabilities to the developer, including migrating entire operating systems to the gateway.

The second virtualization scheme we implemented was Linux Containers (LXC), which is a more lightweight container based virtualization environment. Using LXC, developers can still package their services in a virtualized manner and migrate them to the gateway. In order to implement LXC, the primary requirement of any virtualized application is that it must be supported through the host kernel[1].

We looked at several primary factors to determine which of the two virtualization schemes was the best option, a portion of our results are summarized in Table 1. Three tests are summarized in the table, based on the need for dynamic installation (start time), primary gateway functionality (packet latency), and multitenancy (throughput fairness). To implement a fast start time for the chute, LXC should be used since it boots in 2 seconds compared to 40 seconds for Lguest. We tested packet latency by capturing the time it took for a packet to travel from the Ethernet interface, through the chute, and out the WiFi interface. Compared to the host networking stack, LXC adds minimal latency, around 160μs, whereas Lguest consistently added 39ms to route a packet. The "Throughput Fairness" test demonstrates how fairly each chute can utilize the network bandwidth, which would be required to implement a multitenant based platform. To run this test, we setup two virtual routers based on one of three virtualization methods (none or host, LXC, Lguest) and compared their throughputs, if the throughputs were the same then they shared computational resources perfectly and the result is 50%. In the case where computational resources were not shared fairly, we note those numbers with a fraction of the throughput obtained by each of the two virtual routers (e.g., if a "Host/Lguest" test shows "75%/25%" then the Host achieved 75% of the throughput, leaving Lguest with 25% of the throughput). It should be noted that the "Lguest/Lguest" test resulted in a 30%/30% share of throughput meaning that the computational overhead of Lguest virtualization was enough to cause an overall loss of 40% of the potential throughput of the gateway.

In the end, the overheads of virtualization were too high compared to its benefits, which pushed us to implement LXC over Lguest. While Lguest provided the ability for the developer to implement their own private OS, we determined that the vast majority of services possible would not require kernel level alterations. Likewise, as long as the host operating system supports the kernel module, application, or library, required by the chute, full implementation is possible.

4.2 Resource Management

Once several chutes are operational on the gateway, we needed to determine how well ParaDrop's resource policy management worked. Figure 3 shows key characteristics of two different chutes running on the same gateway. In this test, one chute is streaming a HD video to an end-user's tablet, the other chute is polling a security camera for images and processing them for motion. At 4 seconds into the test, the end-user has requested that the security camera process images as fast as it can (around 22 images per second), since this is a little high for the capabilities of the embedded processor, the performance of the HD video stream degrades slightly. Since the security camera will function properly at a lower frames per second rate, the network operator can request the chute to lower its CPU usage, which is reflected by lowering the image processing rate at 9 seconds into the test. As we can see from the figure, after 10 seconds the HD video stream returns to its normal throughput and the security camera chute is stabilized at a lower processing rate.

[1]ParaDrop is built using the latest OpenWrt version, which uses a 3.8 kernel

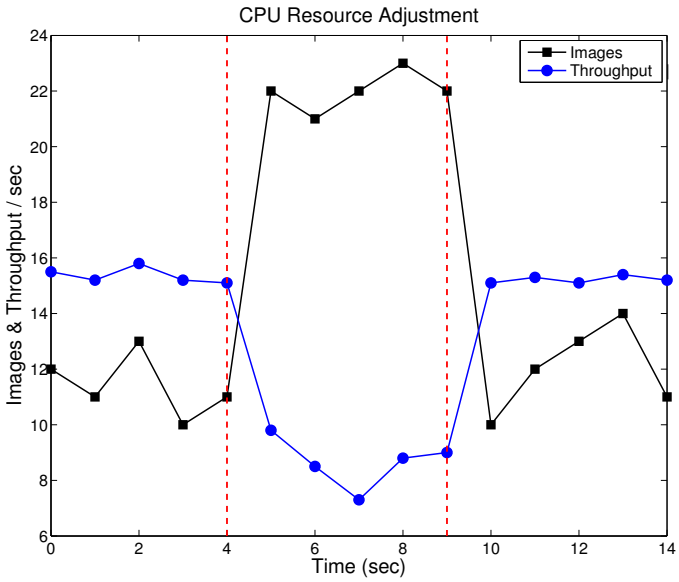

Figure 3: CPU throttle test, shows two chutes competing for resources, the vertical dotted lines represent the user requesting higher security camera CPU utilization and the network operator lowering the utilization.

5. RELATED WORK

Osman[5] introduced the initial work of OS level virtualization through the use of namespace abstraction between the host OS and the VM or Linux Container. The use of Linux Containers as the primary virtualization method has also been seen on other platforms, including Android smartphones with work by Dall[3]. Finally, Soltesz[7] showed that I/O intensive workloads favor Linux Containers as the best virtualization platform when compared to paravirtualization methods such as Xen, due to the much higher efficiency of LXC.

The concept of CPU offloading has seen a recent surge of activity in the technology community. As devices become more heterogeneous and also more mobile, the idea of offloading computational effort, especially to a centralized service such as the cloud looks very appealing. Initial effort in the area, known as Cyber Foraging[1], focused on mobile clients finding computational offloading in the wild through the use of encryption via untrusted hosts.

More recent efforts pushed the virtualization aspects of computing with work like Cloudlets [6]. Using Cloudlets, the focus was on the ability to use centralized services to allow clients to pull computational efforts wrapped in virtualization physically closer to their device.

Finally, the Internet of Things (IoT) has caused a new surge for computation to exist closer to the "edge" of the

network, which as been referred to as Fog Computing[2]. There are several reasons for Fog Computing, including the use of devices as a "gateway service" for wireless sensors, storage, and computation.

These previous works have shown the potential capabilities leveraged by an edge computing framework, whereas our focus remains on the third-party developer's interaction with such a platform. The capabilities provided by ParaDrop allow us to realize the edge computing framework, and expand its potential through a focus on a few key features. These features include the management and installation process of the chutes, as well as the infrastructure realized through our multitenant design.

6. FUTURE WORK

Our design of ParaDrop provides many interesting possibilities and requires multiple directions of future work. For instance, we have established an initial API that needs to be expanded further to allow developers and network operators share the management effort of the gateway, as well as each chute environment. Work towards API standardization has been approached, including merging the API with aspects of existing API paradigms. Projects such as OpenStack would allow developers familiar with that virtualization framework to port already implemented code to the ParaDrop platform. Similarly, significant more work is required in policies of sharing the limited resources of gateways that ensure levels of fairness and function. Our initial work has explored one aspect of this design space and a feedback loop between the framework and third-party services would be desirable. We encourage readers to visit www.paradrop.org to see to continued evolution of the ParaDrop platform over time.

7. ACKNOWLEDGEMENTS

All authors are supported in part by the US National Science Foundation through awards CNS-1040648, CNS-0916955, CNS-0855201, CNS-0747177, CNS-1064944, CNS-1059306, CNS-1345293, CNS-1343363, CNS-1258290, and CNS-1405667.

8. REFERENCES

[1] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 87–92, New York, NY, USA, 2002. ACM.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[3] C. Dall, J. Andrus, A. Van't Hof, O. Laadan, and J. Nieh. The design, implementation, and evaluation of cells: A virtual smartphone architecture. *ACM Trans. Comput. Syst.*, 30(3):9:1–9:31, Aug. 2012.

[4] N. Klingensmith, D. Willis, and S. Banerjee. A distributed energy monitoring and analytics platform and its use cases. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 5:1–5:8, New York, NY, USA, 2013. ACM.

[5] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: A system for migrating computing environments. *SIGOPS Oper. Syst. Rev.*, 36(SI):361–376, Dec. 2002.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, Oct. 2009.

[7] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. *SIGOPS Oper. Syst. Rev.*, 41(3):275–287, Mar. 2007.