ParaDrop: Enabling Lightweight Multi-tenancy at the Network's Extreme Edge

Peng Liu
Department of Computer Sciences
University of Wisconsin-Madison
Email: pengliu@cs.wisc.edu

Dale Willis
Department of Computer Sciences
University of Wisconsin-Madison
Email: dale@cs.wisc.edu

Suman Banerjee
Department of Computer Sciences
University of Wisconsin-Madison
Email: suman@cs.wisc.edu

***Abstract*—We introduce ParaDrop, a specific edge computing platform that provides computing and storage resources at the "extreme" edge of the network allowing third-party developers to flexibly create new types of services. This extreme edge of the network is the WiFi Access Point (AP) or the wireless gateway through which all end-device traffic (personal devices, sensors, etc.) passes through. ParaDrop's focus on WiFi APs also stems from the fact that the WiFi AP has unique contextual knowledge of its end-devices (e.g., proximity, channel characteristics) that are lost as we get deeper into the network. While different variations and implementations of edge computing platforms have been created over the last decade, ParaDrop focuses on specific design issues around how to structure an architecture, a programming interface, and orchestration framework through which such edge computing services can be dynamically created, installed, and revoked. ParaDrop consists of the following three main components: a flexible hosting substrate in the WiFi APs that supports multi-tenancy, a cloud-based backend through which such computations are orchestrated across many ParaDrop APs, and an API through which third-party developers can deploy and manage their computing functions across such different ParaDrop APs.**

We have implemented and deployed the entire ParaDrop framework and, in this paper, describe its overall architecture and our initial experiences using it as an edge computing platform.

I. Introduction

Cloud computing platforms, such as Amazon EC2, Microsoft Azure, and Google App Engine have become a popular approach to providing ubiquitous access to services across different user devices. Third-party developers have come to rely on cloud computing platforms to provide high-quality services to their end-users since they are reliable, always on, and robust. Netflix and Dropbox are examples of popular cloud-based services. Cloud services require developers to host services, applications, and data on offsite datacenters. That means the computing and storage resources are spatially distant from end-users' devices. However, a growing number of high-quality services desire computational tasks to be located nearby. They include needs for lower latency, greater responsiveness, a better end-user experience, and more efficient use of network bandwidth. As a consequence, over the last decade, a number of research efforts have espoused the need and benefits of creating edge computing services that distribute computational functions closer to the client devices, e.g., Cyber Foraging [1], Cloudlets [2], and more recently Fog Computing [3].

This paper presents a specific edge computing framework, ParaDrop, implemented on WiFi Access Points (APs) or other wireless gateways (such as set-top boxes), which allows third-party developers to bring computational functions right into homes and enterprises. The WiFi AP is ubiquitous in homes and enterprises, is always "on" and available, and sits directly in the data path between the Internet and end-user devices. For these reasons, the WiFi AP was selected for ParaDrop as the edge computing platform. ParaDrop uses a lightweight virtualization framework through which third-party developers can create, deploy, and revoke their services in different APs. Their services are inside containers that allow them to retain user state and also move with the users as they change their points of attachment. ParaDrop also recognizes that WiFi APs are likely to be resource limited for many different types of applications, and hence allows for tight resource control through a managed policy design.[1]

The ParaDrop framework has three key components: (i) a virtualization substrate in the WiFi APs that hosts third-party computations in isolated containers (which we call *chutes*); (ii) a cloud backend through which all of the ParaDrop APs and the third-party containers are dynamically installed, instantiated and revoked; and (iii) a developer API through which developers can manage the resources of the platform and monitor the running status of APs and chutes. Compared to other heavyweight counterparts, ParaDrop uses more efficient virtualization technology based on Linux containers [4] rather than the Virtual Machines(VMs), meaning we can provide more resources for services with the given hardware. The virtualization substrate has evolved from LXC [5] to one based on Docker [6]. The change to Docker was due to easy to use tools and wider adoption; this reduces the complexity for new developers to create services on a platform they may be unfamiliar with. We also developed a backend server to manage the gateways through a real-time messaging protocol, Web Application Messaging Protocol (WAMP) [5],

[1]WiFi APs, like all other compute platforms, will continue to get faster and powerful, and the capabilities installable in applications will continue to improve. Our current low-end hardware can perform image processing and motion detection in video feeds.

978-1-5090-3322-5/16 $31.00 © 2016 IEEE
DOI 10.1109/SEC.2016.39

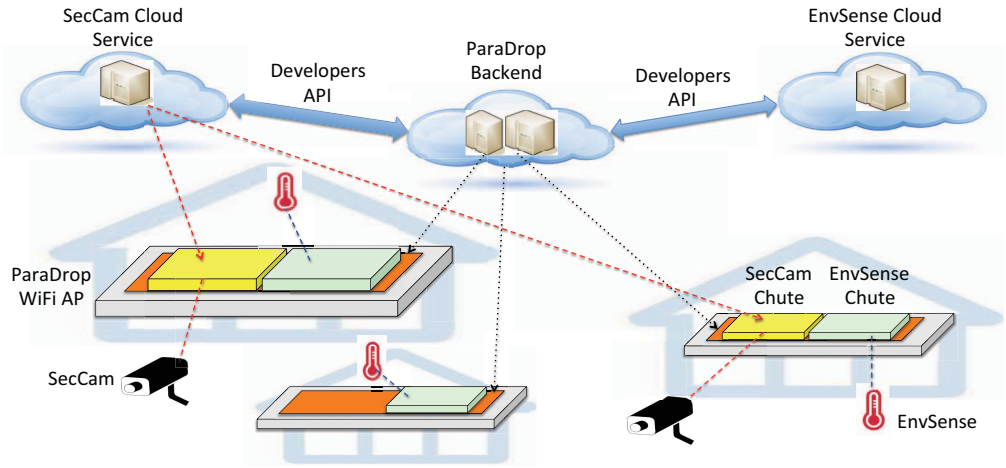

Fig. 1. Overview of ParaDrop. There are three components in the system: the ParaDrop backend, ParaDrop gateways, and the developer API. Developers can deploy services in containers (we call them chutes in ParaDrop, as in parachuting a service on-demand) to the gateways through the backend server. The chutes are deployed using the developer API. Two different chutes are illustrated in this figure: 1) SecCam: a wireless security camera service that collects video from an in-range security camera and does some local processing to detect motions; and 2) EnvSense: a system deployed in buildings to monitor temperature and humidity with sensors. We illustrate these two applications in Section V.

which guarantees very low latency in service deployment and monitoring.

We demonstrate the capabilities of this platform by showing useful third-party applications utilizing the ParaDrop framework. Figure 1 gives an overview of the architecture of ParaDrop and shows two example applications. The first example application is a security camera application that connects with wireless cameras in-range to detect motion. The second application connects to environment sensors in the home, such as temperature and humidity, to determine the position of actuator controlled heating or cooling vents. In addition to the low-latency advantages of ParaDrop, the platform also has unique privacy advantages. A third-party developer can create a ParaDrop service that allows sensitive user data (video camera feed) to reside locally in the ParaDrop AP and not send a continuous feed over the open Internet. Over the last two years, we have installed and deployed ParaDrop as a platform using many different use cases. Our contributions in this paper are the following:

- We discuss the unique challenges in a WiFi AP based edge computing platform with virtualization techniques and propose corresponding solutions to overcome these challenges.
- We propose a system architecture and fully implement it on hardware to provide a reliable and easy to use edge computing platform for developers to deploy edge computing applications.
- We introduce the process to developing applications for ParaDrop with two example applications. We also discuss other possible applications for this platform.
- We analyze and evaluate the system in terms of efficiency and effectiveness. We also discuss the flexibility and scalability.

This paper is organized as follows. First, we give an overview of the goals of ParaDrop in Section II. Then we introduce the design of the platform and some challenges in Section III. Section IV illustrates the implementation of the core components of the system. Section V introduces two example applications we developed for ParaDrop. We also discuss other possible applications that can be deployed on ParaDrop in this section. We evaluate the ParaDrop platform in Section VI. Finally, we discuss related work in Section VII and conclude the paper in Section VIII.

II. ParaDrop Overview

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

With end-user devices going mobile, there is one remaining device that provides the capabilities developers require for their services, as well as, the proximity expected from an edge computing framework. The gateway – which could be a home WiFi AP 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.

We want to push computation onto the home gateways (e.g., WiFi APs and cable set-top boxes) for following reasons:

- The home gateways can handle it. Modern home gateways are much more powerful than what they need to be

for their networking workload. Furthermore, if one is not running a web server out of the home, the gateway sits dormant a majority of the time (e.g., when no one in the home is using it).

- Utilizing computational resources in the home gateway gives us a local footprint within the home for end-devices that are otherwise starved for computational resources. These include sensors such as many inexpensive Internet of Things (IoT) sensors and actuator components. Using ParaDrop, developers can offload some computation of one (or many) IoT devices onto the AP without the need for cloud services or a dedicated desktop. In particular, if one installs different sensors from different vendors, this advantage becomes even more apparent. If a home has controls for blinds, temperature sensors, a specific thermostat, etc., the computational function to make decisions on when to actuate some of these devices are best made, locally, in the ParaDrop AP. They can achieve better efficiency by eliminating unnecessary network traffic, which has additional benefits to avoid network congestion when the network traffic is high.
- Pervasive hardware: Our world is quickly moving towards households only having mobile devices (tablets and laptops) in the home that are not always on or always connected. Developers can no longer rely on pushing software into the home without also developing their own hardware too.

A. A Developer-centric Framework

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. Users (owners or administrators of ParaDrop gateways) can manage and monitor ParaDrop gateways with centralized tools similar to the dashboards for cloud computing platforms. We strive for ParaDrop to be easily deployed and managed by administrators. At the same time, it should be able to improve the user's experience transparently or make specific tasks easy.

B. ParaDrop Capabilities

ParaDrop takes advantage of the fact that resources of a gateway are underutilized most of the time. Each service, referred to as a *chute*, can borrow CPU time, unused memory, and extra disk space from the gateway. This allows vendors an unexplored opportunity to provide added value to their services exploiting the proximity footprint of the gateway.

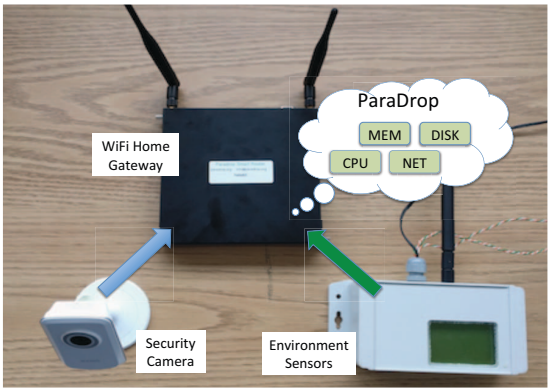

Fig. 2. The fully implemented ParaDrop gateway, which shares its resources with two wireless devices including a Security Camera and an Environmental Sensor.

Figure 2 shows a ParaDrop gateway, along with two services to motivate our platform: *SecCam* and *EnvSense*. The current instance of the ParaDrop gateway has been implemented on a PCEngines single board computer [6] running Snappy Ubuntu on an AMD APU 1GHz processor with 2GB of RAM. By default, the gateway has a 16GB SD card for storage, with the option to use a larger SD card if necessary. We plan to support USB hard drives to provide even larger storage space in the future. This low-end hardware platform was chosen to showcase ParaDrop's capabilities with existing gateway hardware. We have implemented two third-party services by migrating their core functionality to the ParaDrop platform to demonstrate its potential. Each of these services contains 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 security camera based on a commercially available D-Link 931L webcam, which we call *SecCam*. The second service is a wireless environmental sensor designed as part of the Emonix research platform [7], which we refer to as *EnvSense*. Leveraging the ParaDrop platform, the two services allow us to motivate the following advantages of ParaDrop. While some of them generally apply to edge computing platforms, some of the advantages stem from our construction and use of WiFi APs as the hosting substrate. They include:

- *Privacy:* Many sensors and even webcams today rely on the cloud as the only storage mechanism for generated data. By leveraging the ParaDrop platform, the end-user no longer must rely on cloud storage for the data generated by their private devices, but can instead borrow disk space available in the gateway for such data.
- *Low latency:* Many simple processing tasks required by sensors are performed in the cloud today. By moving these simple processing tasks onto gateway hardware which is one hop away from the sensor itself, a reliable low-latency service can be implemented by the developer.
- *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.

- *Local networking:* 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 on the Internet and upon the end-user's request travel back from this server into the end-user's 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's device.
- *Additional wireless context:* A WiFi AP can sense more information about end-devices, such as, the proximity of different devices to each other, location in specific rooms and much more. If such an API is exposed to developers (with proper privacy management), it enables new capabilities complementary to other means of accessing the same information.
- *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.

III. System Design

In this section, we first discuss the challenges to design the ParaDrop system and explain our solutions to overcome these them. We then give an overview of the system architecture and explain the key features to support deploying services on the ParaDrop platform.

A. Challenges And Solutions

ParaDrop uses virtualization technology to provide an isolated environment to services running on the gateway. One key difference of the ParaDrop platform compared to cloud computing platforms is the gateway hardware has lower performance than servers in datacenters. As a result, the efficiency of the virtualization technology is the most important consideration in the system design. There are two types of virtualization approaches: Virtual Machines(VMs) and containers. Hypervisor-based VMs virtualize at the hardware layer, while containers virtualize at the operating system layer. Felter et al. did measurement study to compare the overhead of VMs and Linux containers. More specifically, they compared KVM [8] with Linux containers [4]. They concluded that KVM's complexity makes it not suitable for a workload that is latency-sensitive or has high I/O rates, though both KVM and container have very low overhead on CPU and memory. The VM-based approach provides a fully virtualized environment, which means developers have high flexibility to choose operating systems. However, since it provides access to hardware only, we need to install an operating system in the virtual environment, which quickly gobbles up resources on the gateway, such as RAM, CPU, and disk. Ha et al. proposed an optimized dynamic VM synthesis approach which makes VMs less heavyweight than they appear to be [9]. However, their approach used a very high-performance server as the cloudlet. In contrast to VMs, a container is more lightweight and faster to boot because it does not run a full OS on the virtual hardware. Its high efficiency and low footprint are critical for a low-performance hardware platform. Moreover, we do not think the restriction to use the Linux operating system would be a big concern for developers. Therefore, we choose a container-based virtualization rather than VMs to provide an isolated environment for the services deployed on the gateways. Several management tools are available for Linux containers, including LXC [10], Docker [11], etc.. Due to its feature set and ease of use, Docker has rapidly become the standard management tool and image format for containers [12]. A unique feature of Docker is layered filesystem images, usually implemented by AUFS (Another UnionFS) [13]. The layered filesystem provides the opportunity for us to reduce space usage for the images and simplify the file system management. So we chose Docker to manage the containers in the ParaDrop gateways in the latest version of ParaDrop implementation.

Another challenge encountered while designing ParaDrop is developers normally do not have direct access to the gateways in the deployment because of NATs or firewalls. Therefore, it is hard for them to manage and debug the services running on the gateways. The Virtual Private Network (VPN) is one approach to overcome that challenge. However, with a VPN, the backend server needs to maintain the connections to all the gateways. Developers also need to configure VPN on their machines in order to access the gateways. To simplify the backend server implementation and developers' work, we use the Web Application Messaging Protocol (WAMP) to enable bi-directional communication channels between the ParaDrop backend and gateways. WAMP is an open standard WebSocket subprotocol that provides two application messaging patterns in one unified protocol: Remote Procedure Calls + Publish/Subscribe [5]. With WAMP, we can transmit time-sensitive messages between backend server, developer tools, and gateways with very short latency.

Unlike servers in datacenters, it is hard for us to control the hardware and upgrade software on it after we deploy ParaDrop gateways in the network edge — user's home or office. We need a software system with secure and transactional update capability. The software system also needs to support image backup and rollback features so that we can manage the software running on gateways easily and flexibly. We built the software stack for the gateway based on Snappy Ubuntu [14] which has met our requirements.

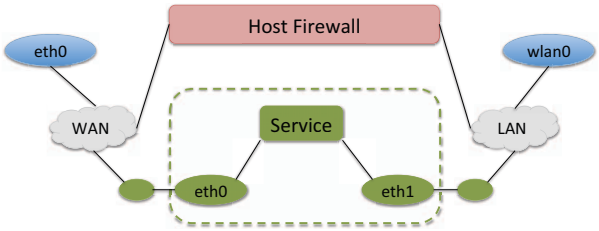

Fig. 3. A chute is running on an AP. The dashed box shows the block diagram representation of a chute installed on a ParaDrop gateway. Each chute hosts a stand-alone service and has its own network subnet.

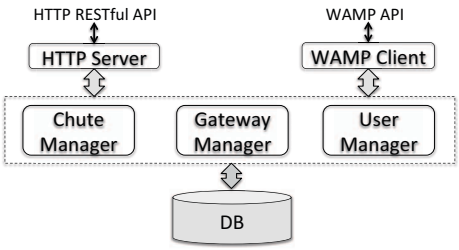

Fig. 4. The architecture of the ParaDrop backend. The backend manages all the resources of the ParaDrop platform and provides APIs for users to deploy services on the gateways.

B. ParaDrop System Architecture Overview

ParaDrop has three core components as shown in Figure 1. The backend and gateways are connected by a WAMP message router. The gateways provide the virtualized environment for the services. Whereas the backend maintains the information of the gateways and users. It also provides a repository (*ChuteStore*) to store files for the services which can be deployed on the gateways. It manages the resource provision to developers and the services running on the routers in a secure way.

C. Implementing Services For The ParaDrop Platform

A service is deployed on the ParaDrop platform in a package called a chute (short for parachute). A user can deploy many chutes (Figure 3) to an AP, thanks to the low-overhead container technology. These chutes allow for isolated use of computational resources on the gateway. As we design and implement services on gateways, we can, and should, separate these services into unique chutes.

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

Dynamic Installation: In order for 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 to ParaDrop. Using the ParaDrop API, the user links the account with their gateways. If a service utilizes a wireless device, the gateway can fully integrate with the service without any interference from the end-user.

ParaDrop API: The focus of ParaDrop is providing high-quality services to end users via third party developers. A seamless RESTful API enables developers 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.

Network Setup: 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 leveraged an SDN paradigm. All configuration related to networking between the chutes and the gateway are handled through a cloud service, which is interfaced by the developers and network operators.

Resource Policies: The multi-tenancy 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, developers specify 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.

IV. Implementation

In this section, we introduce the implementation details of the ParaDrop platform. Though the design motivations and strategies are maintained, ParaDrop has evolved from a VM-based version, to a Linux containers + LXC [4] based architecture, and finally to the current Linux container + Docker [15] based architecture. We only cover the implementation of the latest version of ParaDrop in this paper. As we introduced in Section III, WAMP is used to connect the ParaDrop backend and the gateways. WAMP's two messaging patterns exactly match our requirements to manage and monitor the ParaDrop gateways with minimum latency. We built our system based on Autobahn [16], which provides open source implementations of the WebSocket Protocol and WAMP. We chose crossbar.io as the WAMP router for the ParaDrop platform. Crossbar.io [17] is an open source multi-protocol application router based on Autobahn and WAMP. We installed crossbar.io on an Ubuntu 14.04 machine which is accessible from both the backend server and the gateways.

A. The ParaDrop Backend Implementation

The ParaDrop backend manages the platform resources in a centralized manner. We implemented the ParaDrop backend with the Python programming language based on the Twisted [18] network framework. The major components of the backend are shown in Figure 4. Two important interfaces are implemented for the backend server:

- The WAMP API for the ParaDrop gateways. The backend communicates with all the gateways to dispatch commands, receive responses, and receive status reports. The soft real-time characteristic of WAMP guarantees the minimal latency in the message exchanging [5].
- The HTTP RESTful API for developers, end-users, administrators, and gateways. On the user side, the backend server aggregates the information from all the gateways and provides the information to the ParaDrop frontend. Users can then view the information in a graphical format from the server. The backend server also stores some persistent information about the ParaDrop platform deployment, e.g., the location and configuration of the gateways. Moreover, the backend server relays the commands from users to gateways. Such as starting a chute on one or more specific gateways, and then relaying the responses from the gateways to users.

In addition to the above interfaces, the backend server also contains several authentication mechanisms. To get permissions for a resource, all users need to register for an account on the backend server. The backend server stores information about the users, gateways, and chutes in a MongoDB database [19]. At this time, we are in the process of implementing a repository to manage the published chutes submitted to the backend server. Users will be able to deploy chutes from that repository to their gateways through the web frontend. Currently, in order to deploy a chute to a gateway, the chute package must be installed using the ParaDrop developer console on a local machine.

B. The ParaDrop Gateway Implementation

The ParaDrop gateway is the execution engine of the ParaDrop platform. First of all, we need to securely and reliably maintain the ParaDrop software components on the operating system of a gateway. In Snappy Ubuntu, every software component is a Snappy package, which allows transactional upgrades or rollbacks of the component reliably [14]. To that end, we chose Snappy Ubuntu as the operating system.

On the one hand, we want to provide an isolated and virtualized environment for the chutes deployed on a gateway. On the other hand, we want to keep the virtualization overhead as low as possible because the hardware platform of the gateway is not as powerful as a server in cloud computing platforms. These are just a couple of reasons why we chose Linux containers and Docker to build the virtualized environment for the chutes. Docker has features such as layered image and NAT, which ease the development, management, and deployment of chutes. Docker also effectively solves the dependency problem when developers migrate software tested in a development environment to a deployment environment. Since Docker is an App-container, it is perfect for a chute that only requires one process. If a complex service needs to run multiple processes, we can either launch multiple containers or use the Docker supervisor. Also, since Docker is a pure execution environment, we need to manage the networking environment in our implementation.

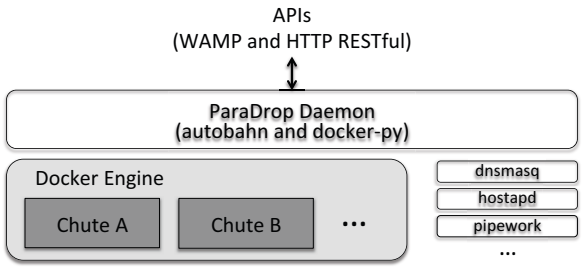

Fig. 5. Major software components on a ParaDrop gateway. All the components are snappy packages. We implemented the ParaDrop daemon with Python from scratch. And we made the snappy packages for dnsmasq and hostapd based on the corresponding open source projects. The Docker engine is packaged by Ubuntu. All the snappy packages can be securely and transactionally updated over-the-air if required.

A ParaDrop daemon (also called Instance tools) runs on the gateway to implement all the functions required by the ParaDrop platform. It implements all Docker-related features based on a Python wrapper of Docker APIs. Additionally, it exposes the controlling and monitoring interfaces to outside world. Its major responsibilities include:

- Registers the gateway to the ParaDrop backend.
- Monitors the gateway's status and reports it to the ParaDrop backend.
- Manages resources and processes including virtual wireless devices, firewall, DHCP, hostapd, etc., to provide an environment for the chutes managed by the Docker engine and also for the devices connecting to the gateway.
- Receives RPCs and messages from the ParaDrop backend and manages containers on the gateway accordingly, e.g., install, start, stop, and uninstall chutes.

The ParaDrop daemon interfaces support both WAMP and HTTP. Therefore, a ParaDrop gateway can be accessed by the developer tools indirectly through the WAMP message router or directly if they are on the same network. Every chute must define the resource requirements in a config file, as the ParaDrop daemon enforces resource policies on all installed chutes. These policies are enforced when it is creating or starting a chute instance. ParaDrop supports enforcement for the resources listed below:

- CPU: CPU resource for a chute is controlled by a *share* value of the container. We can specify the share value when we create a container, or change it on-the-fly when a container is running. The *share* value defines relative share of the CPU resource that one chute can use when it competes with other chutes. It does not limit the maximum CPU resource a chute can use. We explain that in detail in Section VI.
- Memory: The maximum memory size can be used by a chute is restricted by a value defined in its config file.
- Network: The ParaDrop daemon tracks all the network interfaces used by chutes and restricts their bandwidth by traffic shaping. We may implement a strategy based on *share* values similar to the CPU resource management

in a future release.

Some edge computing applications require the storage resource to cache content, or to provide local storage services to users. Hence, we need a flexible way to manage the limited storage resources in the gateways. We plan to implement policies to manage the block devices based on the Linux kernel's device mapper and Docker's *devicemapper* storage driver [20]. Example policies include disk size quota for a chute, read/write speed. Currently, we only define a common maximum disk size (1GB) for all chutes.

C. The ParaDrop Developer Console Implementation

We implemented the developer console with Python. Since a chute can be implemented with any languages or libraries, the developer console is agnostic to the contents of chutes. Essentially a chute is a Docker image with ParaDrop-specific files. The developer console provides the tools that we can use to interact with the ParaDrop platform. Developers can create chutes locally, upload chutes to the backend and install chutes from local machine or backend to the gateways if they have permissions to do so. We can also monitor the status of the chutes and gateways. If a developer has direct access to the gateways (in development environment), he or she can access the gateways directly without the WAMP router.

D. The ParaDrop Web Frontend Implementation

The ParaDrop platform has a Web frontend. It provides all the features of the developer console except building a chute. Moreover, it provides a better user interface for users and administrators to install chutes on their gateways in an intuitive way. The web interface to monitor the status of chutes and gateways is also nicer than the command line tools.

E. The ParaDrop Workflow

To develop an application for the ParaDrop platform, a developer needs to interact with two components:

- The build tools in the developer's computer.
- The instance tools in the ParaDrop daemon.

Figure 6 shows the two components that a developer needs to work with. By providing tools to create and manage chutes, the platform implementation is transparent to developers. They can view the ParaDrop hardware as familiar resources close to the users mobile devices and focus on the application logic development.

It is recommended a developer test the application locally, and then pack the binary with the configuration files into a chute with the build tools. Upon successful chute packing, testing with the local ParaDrop gateway should follow. Once all the bugs and functional verification tests have been completed, the developer can either install the chute to the routers or publish the chute to the *ChuteStore*.

V. ParaDrop Applications

Developers can deploy various chutes (applications and services) on the ParaDrop platform. Any service currently running on a cloud computing platform that can take advantage

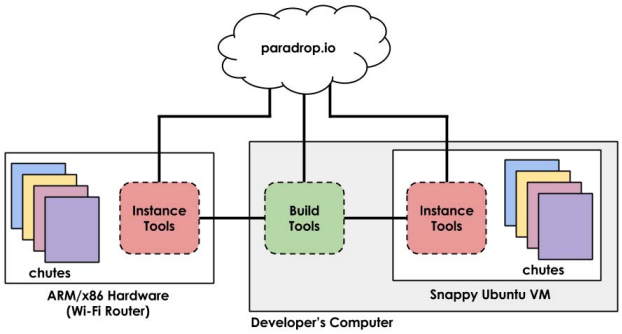

Fig. 6. The ParaDrop platform from a developer's view: We refer to Build Tools when we talk about the command line program running on developer's development computer that control and communicates with the rest of the ParaDrop platform. The Instance Tools leverage Docker to allow ParaDrop apps to run on router hardware. This "hardware" could be a ParaDrop gateway, a Raspberry Pi, or even a virtual machine on your computer that acts as a router (which is why we call it an Instance sometimes).

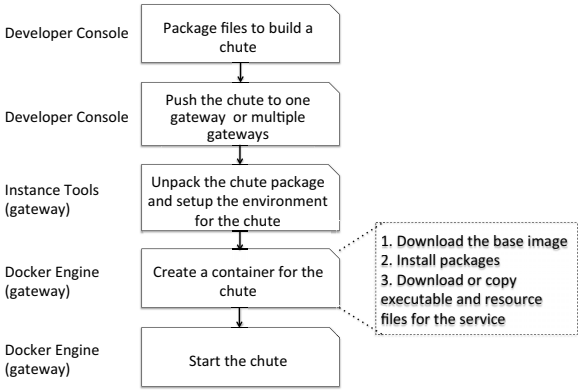

Fig. 7. The process to build, deploy and launch a chute on a gateway. The step to create the container for the chute could be simplified by building the Docker images and uploading them to a private Docker repository beforehand.

of the edge computing resources offered by the ParaDrop platform can be easily ported with just a few changes. In order to make the service compatible with the ParaDrop platform, we need to provide some configuration files to define the resource requirement of the service, e.g., CPU, memory, network. With the ParaDrop developer tools, we can package the binary files, scripts, Dockerfile, and ParaDrop configuration files to a chute. The chute can then be deployed on ParaDrop gateways to provide the service. Figure 7 illustrates the process to deploy and launch a chute on a gateway. Depending on the applications requirements, we can either deploy the whole service to the ParaDrop platform; or for a complex service composed of many microservices, we can deploy parts of the service (some microservices) to the ParaDrop platform. In the latter case, some microservices running in the cloud collaborate with the microservices running on the ParaDrop platform to provide the service to end-users.

In this paper, we present two IoT example applications in detail. IoT is becoming a huge part of the networking

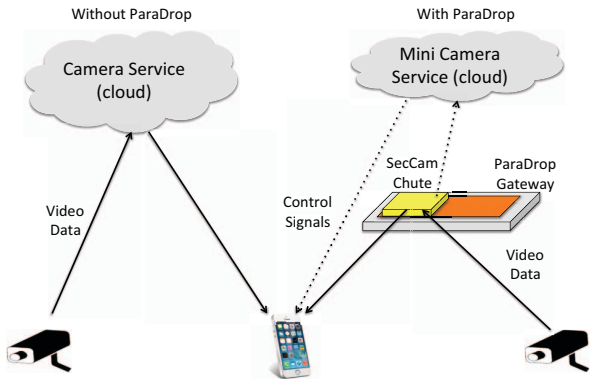

Fig. 8. The IP camera service without ParaDrop and with ParaDrop. With ParaDrop, we can deploy the camera service into the gateway. If the mobile device has a direct connection with the camera service on the gateway, it will control the camera and get video data from it directly without relying on the service deployed in the cloud. If not, the camera service on the gateway still can do most work (e.g. motion detection) and it can create a peer-to-peer connection to the mobile device with the help of the mini camera service in the cloud. We do not discuss the mini camera service in this paper for brevity. ParaDrop makes the chute deployment easy and flexible.

world. Yet, many IoT devices rely on backend services that must traverse the Internet to utilize their full potential. Using ParaDrop, we can pull that intelligence into the edge — the gateways.

A. The SecCam Service

In this section, we present a walkthrough about using a wireless video camera with a ParaDrop gateway to implement a security camera service called SecCam. The SecCam service is based on a commercially available wireless IP camera (D-Link 931L webcam), where we took the role of developer to fully implement the service.

D-Link provides a cloud service mydlink [21] to which users can register their camera and can access the camera remotely. They can view the image and modify the settings with a web client or mobile apps supporting "mydlink" cloud service. The cloud service is very convenient to setup, but the video data need to be pushed to the cloud platform and users do not have much control over it. We propose to move the functionality of the cloud service to the ParaDrop gateway to give users full control of the camera device and the video data.

For this service, we require network interfaces to communicate with the webcam and the Internet, as well as ample storage for images. Figure 8 compares the camera service deployed in cloud and ParaDrop.

We need to create a chute for the *SecCam* service, which is responsible for:

- **Creating the SecCam SSID.** This SSID provides an isolated WiFi network and subnet to the security cameras. The devices purchased by end-users do not have to be programmed when they arrive at the purchaser's home (they can be flashed with a default SSID and password by the company).

```
owner: ParaDrop
date: 2016-02-06
name: SecCam
Description:
   This app launches a virtual wifi AP |
   and detects motion on a wifi webcam.
net:
   wifi:
      type: wifi
      intfName: wlan0
      ssid: seccamv2
      key: paradrop
      dhcp:
         lease: 12h
         start: 100
         limit: 50
resource:
   cpu: 1024
   memory: 128M
dockerfile:
   local: Dockerfile
```

Fig. 9. The configuration file of the SecCam chute. It defines the environment of the container to run the chute.

```
FROM ubuntu:14.04
MAINTAINER ParaDrop Team <info@paradrop.io>

RUN apt-get update && apt-get install -y \
   apache2 \
   libapache2-mod-php5 \
   python-imaging

ADD http://paradrop.io/storage/seccam/srv.tar.gz /var/www/
RUN tar xzf /var/www/srv.tar.gz -C /var/www/html/
ADD http://paradrop.io/storage/seccam/cmd.sh /usr/local/bin
CMD ["/bin/bash","/usr/local/bin/cmd.sh"]
```

Fig. 10. The Dockerfile of the SecCam chute. This Dockerfile is a simplified version. We need to download some files for the chute from a web server. In deployment, these files can be included in the chute package.

- **Image capture service.** This is a Python-based program with user-defined characteristics such as a threshold of motion, time of day, and rate of detection. The service captures images from an IP camera, calculates differences to detect motion, and stores those images to disk. The images stored on disk will then be visualized using a web server which runs inside the chute.
- **Web server.** The web server is the mechanism that allows the end-users on the local network to view live video, check the logs, and view the motion detection images stored on the ParaDrop device.

A chute is described by a ParaDrop configuration file and a Dockerfile.

- The ParaDrop configuration file is a YAML [22] formatted file for ParaDrop. It describes the resource and environment requirements of the chute.
- The Dockerfile follows the specification of Docker. It defines the Docker image which is managed by the Docker engine on the ParaDrop gateways.

Figure 9 shows the configuration file of the SecCam chute and Figure 10 shows its Dockerfile. The Docker image is based on Ubuntu 14.04. The static file of the web server and the source code of image capture service will be installed on the image when we launch the chute on a ParaDrop gateway. Alternatively, we can store the pre-built image in a private Docker repository so that we can download and launch the

chute directly.

B. The EnvSense Service

This service is a wireless environmental sensor designed as a part of the Emonix research platform [7]. Since the service is fully implemented, we only need to migrate the service to fit the ParaDrop platform, rather than rewrite it from scratch. The original service runs on a server to collect data from the sensors, process, store and visualized the data. After identifying the resources required to run the service, we can create a chute for it. The steps to create the configuration file are similar to the SecCam service so we omit them here for brevity.

We can divide the EnvSense service to multiple microservices. For example, we can run data collection and processing microservices on the ParaDrop platform and run data storage and visualization microservices on the cloud platform. It will be fairly easy with the development and management tools provided by ParaDrop to implement that change.

C. Other Possible Applications

Other than IoT, applications in other categories can also take advantage of the ParaDrop platform. For instance, Peer-to-Peer (P2P) technology is a popular approach to reducing the workload of the media server [23]. However, it is challenging to use P2P on mobile devices because of the following issues:

- Mobile devices are powered by batteries so that they can not afford the overhead to upload the media data or share the data with other peers.
- The wireless connections have restricted bandwidth compared to wired connections, in many cases, mobile devices do not have abundant bandwidth can be used to share media data with other peers.
- Compared to a wired network, a wireless network is dynamic, so the overhead to maintain the status of peers can be too high and offset the potential advantages of P2P technology.

A WiFi gateway, as the last hop of mobile devices connection to the media server, is a good place to deploy the P2P technology. With ParaDrop, they can be implemented in a chute in order to provide transparent service to the mobile devices.

Another type of application is a caching service. We built a media caching service on the ParaDrop platform to prefetch and cache video data from Netflix. That service can improve user experience by eliminating network congestions.

VI. System Evaluation

In this section, we discuss the evolution of the ParaDrop platform and evaluate the system regarding efficiency and effectiveness. We also discuss the system's usability and scalability.

TABLE I
Overhead of the virtualization schemes on the first generation hardware platform of ParaDrop gateway

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

A. Efficiency Of Virtualization

The selection of a virtualization solution is the core of the ParaDrop platform. The platform itself has already been through three generations of virtualization schemes throughout its evolution.

- **Hypervisor-based virtualization: OpenWRT [24] + lguest.** In the first generation of the ParaDrop platform, we implemented the virtualization solution based on lguest. Lguest is a small x86 32-bit Linux hypervisor for running Linux under Linux [25]. A number of hypervisor solutions have appeared that use Linux as the core, including KVM [8] and lguest. We selected lguest for our first generation hardware platform, because of its relatively simple implementation (5000 lines of code) and availability in the mainline Linux kernel. Though the simplicity of lguest is attractive, it is slower than other hypervisors. A recent measurement study by Felter et al. confirmed that VMs have high latency in I/O operations [12] and they are not suitable to latency sensitive applications. Another problem of lguest is it is a paravirtualization hypervisor, which means the guest OSes need to be lguest-enabled. This increases the complexity when porting services and creates a barrier for developers to go through in adopting the platform.
- **Linux Containers: OpenWRT + LXC.** Looking for a more lightweight virtualization scheme, we migrated to the LXC (Linux container) management tool. Containers have lower overhead than hypervisor-based virtualization solutions, because they share the same operating system and do not emulate hardware. There are two user-space implementations of Linux containers, each exploiting the same kernel features [10]. We selected LXC because it is flexible and has more userspace tools. In order to implement container-based virtualization, the application must be supported by the host Linux kernel. We did not think that would be a big concern for many applications. Dale et al. measured the overhead of the first two generations of virtualization schemes [26] and the results are shown in Table I. Lguest needs much longer time to start because it needs to boot the guest operating system before it can run an application. It should be noted, the result regarding packet latency in Table I is not consistent with the results of KVM reported by Felter et al. [12]. Table I also shows the throughput fairness, LXC can achieve good fairness along with much lower overhead than lguest.

TABLE II
CHUTE OPERATIONS BENCHMARK ON THE LATEST HARDWARE PLATFORM OF PARADROP GATEWAY

Operation	Time (sec)
Deploy	527
Start	5
Stop	17
Delete	7

- **Linux Containers: Snappy Ubuntu + Docker.** Although the second generation virtualization software had very low overhead and the implementation on OpenWRT is quite efficient for the hardware, we began to realize the operating system disparity on the gateway and the cloud platform could be an obstacle to application development. Upon this realization, we migrated to Snappy Ubuntu and upgraded the hardware to a 64-bit processor platform. This was a significant change and caused many software architecture modifications in the process of going from not only an OS alteration, but also a virtualization modification. For the record, Docker has similar performance on start time, packet latency and throughput fairness as shown in Table I for LXC.
  To have a clear idea about the latency to deploy, start, stop and delete a chute on a gateway, we measured the time taken for these operations on the latest hardware platform. The results are shown in Table II. We tested the system with the *SecCam* chute we described in Section V. The test results depend on the network bandwidth because we need to download an Ubuntu 14.04 base image from the Docker repository, some Ubuntu packages from an Ubuntu repository and a package including all the files for the chute from a file server. We are optimizing the ParaDrop daemon implementation to reduce the time taken for these operations. The chute deployment takes very long time. Figure 11 illustrates the time taken by each step. Fortunately, the ParaDrop gateway can cache the images so that we do not need to download and configure an image to deploy a chute if the chute can share the base image with another deployed chute.

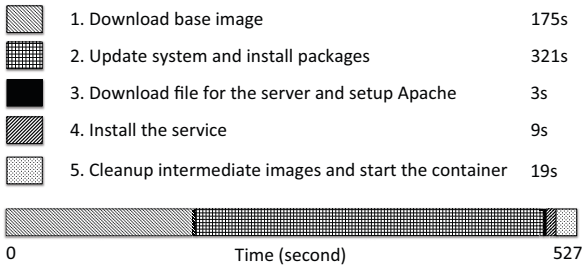

Fig. 11. Time taken by each step in a chute deployment. Step 2 took 321 seconds, which is more than 50% of the total time. If an application is sensitive to deploying time, we can pre-build and store the image for that application in the private repository, then step 2,3,4 and a large part of step 5 can be eliminated. So that we only need to download the image and start the container to deploy a chute.

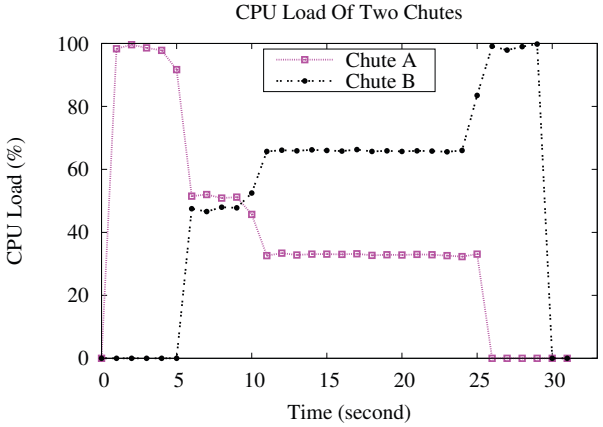

Fig. 12. The CPU resource management test. We started two chutes from the beginning. Chute A and B have *share* values 512 and 1024 respectively. Both of them can stress test the CPU when they work. Chute A started working from the beginning and became idle after 25 seconds. Chute B started working 5 seconds later, it also kept working for 25 seconds then became idle. In the beginning, chute A's CPU load is about 99%. After 5 seconds, Chute A and B started to compete for CPU resource. They achieved a balance in 11 seconds and Chute B's CPU load was about twice of Chute A's CPU load since then. After chute A became idle, chute B used all the CPU resource and the CPU load was about 99%. Total CPU load of the containers is always about 99% because there was no other computationally intensive task running on the gateway.

B. Effectiveness Of Resource Management

To evaluate the effectiveness of the resource management of the ParaDrop platform, we developed two test chutes and deployed them on a ParaDrop AP. Figure 12 shows the result. Docker uses cgroups to group processes running in a container and allows us to manage the resources for containers. When we build a container, we can specify the CPU *share*. The default value is 1024. This value does not mean anything when we talk about it alone. But when two containers both want to use 100% CPU, the *share* values will decide how much share of the CPU that the containers can use. For example, if container A's *share* is 512, container B's *share* is 1024. When two containers are busy running in the same time, container B will have two times share than container A. However, Docker does not limit a container to use free resources. For example, if container B is idle, then container A can use all available CPU resources. The test results indicate the container's CPU resource can be effective managed. Currently, we can specify the CPU *share* value for the chute in the configuration file. It is also possible to change the CPU *share* of the container on-the-fly, though we do not provide this API for now.

By default, every chute can use all of the memory resources available in the gateway. That is not encouraged because it can lead to issues that one chute can easily make the system unstable by allocating too much memory. We can specify the maximum memory that a chute can use in the configuration file. One thing we should note is we can also control the swap available for a chute. By default we disable the swap usage

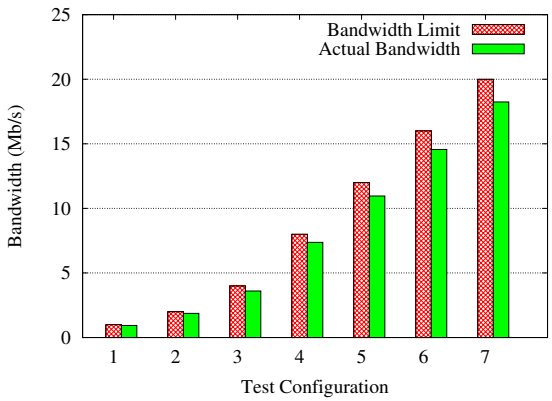

Fig. 13. Network speed limit test. We launched a test chute including an HTTP server and a 100MB file. In order to avoid the network bandwidth variation of wireless interfaces, we conducted the test with an Ethernet interface. We limited the network bandwidth of the chute to seven different values and tested the actual bandwidth when a user download the large file from the chute. Without the speed limit, the network bandwidth is 89.6Mb/s. We can see the actual bandwidths are not higher than the limits and the discrepancy is small.

for the chute, so we need to be careful to specify enough (but not too much) memory to a chute. We verified that with a test chute which allocates 256MB memory. When we limited the memory allocated to the chute is only 200MB, the chute could not start successfully. It would start successfully when the limit is 260MB though.

The network is an important resource for the services deployed on a gateway. All the running chutes share the network bandwidth (LAN and WAN side). Docker (the version we are using) does not provide the support to throttle bandwidth. So we employed the traffic shaping feature of the *tc* command to implement the network resource management for chutes. For the chutes that need network interfaces, we use the *tc* command to limit the bandwidth of these network interfaces for these chutes. Test results in figure 13 indicate our approach is effective.

A ParaDrop gateway has an SD card with 16GB capacity in a typical configuration. By default every container in the Docker can get 10GB of space, that is too much for a chute to run on a ParaDrop gateway. We changed that to be 1GB for every chute, and that should be enough for most applications. In the future, we plan to support different quotas for different chutes.

The read/write performance of the SD card is not very high. We need to control the speed that a chute access the file system or else one chute would slow down all other chutes easily. That remains a future work.

C. Flexibility And Convenience To Deploy Services In The Edge

ParaDrop is highly flexible in deploying edge computing applications. This flexibility is guaranteed by the following design choices:

- The underlying operating system on the ParaDrop AP is Snappy Ubuntu. It provides secure and transactional update capabilities to reliably manage the software on the gateways. Furthermore, cloud computing platforms operating systems are quite similar to Snappy Ubuntu. Developers familiar with these systems can easily port over or develop on the ParaDrop platform.
- We implemented a virtualized environment based on Docker for the gateway. Developers can select the programming languages, libraries, and platforms based on their experience and requirements. For example, a developer can use different versions of Python to develop the services. That flexibility makes the ParaDrop platform easily acceptable to a vast number of developers. In addition, Docker is a popular virtualization solution for cloud computing service deployment. Therefore, the barrier to port a whole service or some microservices of a large service from a cloud computing platform to the ParaDrop platform is very low. The only requirement for services on the ParaDrop platform is that they should be able to run on a relatively new version of the Linux kernel. We believe that will not be a problem in practice. Compared to developing an application for Android or iOS, developers utilizing the ParaDrop platform do not need to install SDKs or learn new application frameworks.
- The WAMP-based messaging simplified the deployment of the ParaDrop gateways. ParaDrop leverages both the remote procedure call and publish/subscribe messaging schemes to implement the communications between the ParaDrop backend server and ParaDrop gateways. Developers do not need to maintain direct connections to the gateways to debug their chutes in deployment.

D. Scalability

By splitting and distributing the services on the cloud to the ParaDrop gateways, ParaDrop provides a strong platform to deploy services at large scales. We only transmit latency-sensitive messages with WAMP and transmit other messages with HTTP. We do this so that the WAMP router (crossbar.io) is not the bottleneck of the system even with a large number of APs. The web server in the ParaDrop backend can be replicated if necessary to support large-scale deployment.

VII. Related Work

Virtualization. Virtualization is the core technology in cloud computing. Both academia and industry have explored different approaches to virtualizing the hardware resources. The idea of virtualization is not new. The origins of the technology can be traced back to the ages of the mainframe. But in the last decade, the fast development and applications of cloud computing lead to a fast evolution of virtualization technologies. Virtualization is widely used to improve the resource utilization and to simplify the datacenter management. Hypervisor-based virtualizations, e.g., Xen [27], VMware ESX [28], KVM [8] and lguest [25], emulate hardware resources to the guest operating system and they can achieve high

flexibility. However, they have drawbacks on booting time and overhead. Container-based virtualization, also known as operating system (OS) level virtualization, is an alternative technique, and it has the advantage on efficiency over hypervisor-based virtualization. Examples include FreeBSD jails [29] and Solaris Containers [30]. Because of the availability of supporting technologies in the Linux kernel, like namespace [31] and cgroups, container-based virtualization is becoming a popular approach for the Linux operating system [32], [4]. In addition to the applications on cloud computing in datacenters, researchers predicted the virtualization would be widely used in other environments, such as desktops and smartphones [33]. ParaDrop leverages the virtualization technology to provide an isolated and controlled environment to the services deployed at the network edge. We are using it in a distributed, resource-restricted, environment. Whereas cloud computing uses virtualization technology in a closed and managed system with high-performance hardware.

CPU offloading. CPU offloading approaches propose to offload computationally intensive tasks from battery-driven mobile devices to co-located specialized devices or cloud to improve the battery life or performance of mobile devices. MAUI [34] and ThinkAir [35] are two examples that enabled fine-grained computing offload of mobile code to the infrastructure. Rather than offload tasks from the mobile devices to the cloud, ParaDrop deploys services or microservices in the wireless gateways to support applications on mobile devices.

Edge computing. Many researchers have explored the advantages of edge computing and proposed different approaches to using them. Balan et al. proposed cyber foraging: a mechanism to augment the computational and storage capabilities of mobile devices. Cyber foraging uses opportunistically discovered servers to improve the performance of interactive applications and distributed file system on mobile clients [1]. Satyanarayanan et al. proposed cloudlet, a trusted, resource-rich computer or cluster of computers that's well-connected to the Internet and available for use by nearby mobile devices [2]. Cloudlet can achieve interactive response because of the cloudlet's physical proximity and one-hop network latency. Bonomi et al. discussed Fog Computing, which brings data processing, networking, storage and analytics closer to mobile devices. They argued that the characteristics of Fog Computing, e.g., low latency and location awareness, very large number of nodes, make it the appropriate platform for many critical Internet of Things services and applications [3].

Some applications were built to leverage the advantages of edge computing architecture. MOCHA is a mobile-cloudlet-cloud architecture that partitions tasks from mobile devices to the cloud and distribute compute load among cloud servers (cloudlet) to minimize the response time [36]. Ha et al. describe the architecture and prototype implementation of an assistive system based on Google Glass devices [37]. Zhang et al. built Vigil, a real-time distributed wireless surveillance system that leverage edge computing to support real-time tracking and surveillance in different scenarios [38]. They use very powerful machines to build the cloudlet. However, ParaDrop uses the hardware platform with medium resources because it has different goals. The targeting applications of ParaDrop are those requiring low latency and resource always on and available.

Smart routers. As the performance of WiFi routers keeps increasing, many companies are interested in building smart routers that can be managed and monitored with mobile Apps [39], [40]. Users can even install third-party applications on some of them [41]. Their goal is to optimize the experience of mobile devices users, whereas ParaDrop's goal is to push services from datacenters to the network edge.

VIII. Conclusion and Future Work

ParaDrop is a flexible edge computing platform that users can deploy diverse applications and services at the *extreme* edge of the network. In this paper, we introduce the three key components of the platform: a flexible hosting substrate in the WiFi APs that supports multi-tenancy, a cloud-based backend through which such computations are orchestrated across many ParaDrop APs and an API through which third party developers can deploy and manage their computing functions across such different ParaDrop APs. We built the system with low-overhead virtualization technology to efficiently use the hardware resources of the ParaDrop APs. We implemented effective resource management policies to provide a controlled environment for services running on the ParaDrop APs.

We have already conducted tutorials and workshops with ParaDrop in multiple forums (at the US Ignite 2014 conference and a GENI Engineering Conference also in 2014) with great success. Users were able to build services such as SecCam from scratch, within a few hours, providing some preliminary evidence of its ease of use.

The platform is still under active development. We will continue to evolve the APIs. We plan to add more accessories support to the ParaDrop gateway, e.g., Bluetooth Low Energy (BLE) and audio sensors, to improve the capabilities of the gateways further. We are in the progress to implement the *ChuteStore* for developers to publish their chutes and for users to search and install chutes on their wireless gateways.

IX. Acknowledgements

We thank Mickey Barboi, Sejal Chauhan, Lance Hartung, Derek Meyer, and other students for their help in the system development and deployment. We are grateful to our shepherd, Sape Mullender, and the anonymous reviewers whose comments helped bring the paper to its final form. All authors are supported in part by the US National Science Foundation through awards CNS-1555426, CNS-1525586, CNS-1405667, CNS-1345293, and CNS-1343363.

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*. ACM, 2002, pp. 87–92.

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

[3] 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*. ACM, 2012, pp. 13–16.

[4] (2016) Linuxcontainers.org, infrastructure for container projects. [Online]. Available: https://linuxcontainers.org/

[5] (2016) Wamp: The Web Application Messaging Protocol. [Online]. Available: http://wamp-proto.org/

[6] (2016) PC Engines apu platform. [Online]. Available: http://www.pcengines.ch/apu.htm

[7] N. Klingensmith, J. Bomber, and S. Banerjee, "Hot, cold and in between: enabling fine-grained environmental control in homes for efficiency and comfort," in *Proceedings of the 5th international conference on Future energy systems*. ACM, 2014, pp. 123–132.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the linux virtual machine monitor," in *Proceedings of the Linux symposium*, vol. 1, 2007, pp. 225–230.

[9] K. Ha, P. Pillai, W. Richter, Y. Abe, and M. Satyanarayanan, "Just-in-time provisioning for cyber foraging," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 2013, pp. 153–166.

[10] (2016) Ubuntu Documentation - LXC. [Online]. Available: https://help.ubuntu.com/lts/serverguide/lxc.html

[11] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[12] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015, pp. 171–172.

[13] (2016) Aufs4 – advanced multi layered unification filesystem version 4.x. [Online]. Available: http://aufs.sf.net

[14] (2016) Snappy Ubuntu. [Online]. Available: https://developer.ubuntu.com/en/snappy/

[15] (2016) Docker homepage. [Online]. Available: https://www.docker.com/

[16] (2016) Autobahn: Open-source real-time framework for Web, Mobile and Internet of Things. [Online]. Available: http://autobahn.ws/

[17] (2016) Crossbar.io: Connecting Systems, Devices and People. [Online]. Available: http://crossbar.io/

[18] (2016) Twisted matrix labs: Building the engine of your Internet. [Online]. Available: https://twistedmatrix.com/trac/

[19] K. Chodorow, *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[20] (2016) Docker and the Device Mapper storage driver. [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/

[21] (2016) mydlink. [Online]. Available: https://www.mydlink.com/entrance

[22] (2016) YAML: YAML Ain't Markup Language. [Online]. Available: http://yaml.org/

[23] Y. Liu, Y. Guo, and C. Liang, "A survey on peer-to-peer video streaming systems," *Peer-to-peer Networking and Applications*, vol. 1, no. 1, pp. 18–28, 2008.

[24] F. Fainelli, "The openwrt embedded development framework," in *Proceedings of the Free and Open Source Software Developers European Meeting*, 2008.

[25] R. Russel, "lguest: Implementing the little linux hypervisor," *OLS*, vol. 7, pp. 173–178, 2007.

[26] D. Willis, A. Dasgupta, and S. Banerjee, "Paradrop: a multi-tenant platform to dynamically install third party services on wireless gateways," in *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*. ACM, 2014, pp. 43–48.

[27] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 164–177, 2003.

[28] A. Muller and S. Wilson, "Virtualization with vmware esx server," 2005.

[29] M. K. McKusick, G. V. Neville-Neil, and R. N. Watson, *The design and implementation of the FreeBSD operating system*. Pearson Education, 2014.

[30] M. Lageman and S. C. Solutions, "Solaris containerswhat they are and how to use them," *Sun BluePrints OnLine*, pp. 819–2679, 2005.

[31] R. Pike, D. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, "The use of name spaces in plan 9," in *Proceedings of the 5th workshop on ACM SIGOPS European workshop: Models and paradigms for distributed systems structuring*. ACM, 1992, pp. 1–5.

[32] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, "Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 275–287.

[33] K. L. Kroeker, "The evolution of virtualization," *Communications of the ACM*, vol. 52, no. 3, pp. 18–20, 2009.

[34] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[35] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *INFOCOM, 2012 Proceedings IEEE*. IEEE, 2012, pp. 945–953.

[36] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-vision: Real-time face recognition using a mobile-cloudlet-cloud acceleration architecture," in *Computers and Communications (ISCC), 2012 IEEE Symposium on*. IEEE, 2012, pp. 000 059–000 066.

[37] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 68–81.

[38] T. Zhang, A. Chowdhery, P. V. Bahl, K. Jamieson, and S. Banerjee, "The design and implementation of a wireless video surveillance system," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 426–438.

[39] (2016) Smart wifi app center. [Online]. Available: http://www.linksys.com/us/smart_wifi_center

[40] (2016) Meet OnHub. A new type of router for the new way to Wi-Fi. [Online]. Available: https://on.google.com/hub/

[41] (2016) HiWiFi Apps. [Online]. Available: http://www.hiwifi.com/j3-func