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 (modest) 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.) pass through. Paradrop's focus on the 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 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 own 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 some of our initial experiences in 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 provide 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 requires developers to host services, applications and data on offsite datacenters. That means the computing and storage resources are relatively far away from end-users' devices. But, due to application-specific reasons, 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 a unique platform for edge computing because of multiple reasons: it is ubiquitous in homes and enterprises, is always "on" and available, and sits directly in the data path between Internet resources and the end device. Paradrop uses a lightweight virtualization framework through which such third-party developers can create, deploy, and revoke their services in different APs inside compute 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 multiple key components: (i) a virtualization substrate in the WiFi APs that host 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 such 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), that means we can provide more resources for services with the given hardware. Our virtualization substrate evolved from our original choice of LXC [5] to one based on Docker [6] because of the latter's easy to use tools and wider adoption that would reduce the bar for developers. We also developed a backend server to manage the gateways, and the communication between them is through a real-time messaging protocol — Web Application Messaging Protocol (WAMP) [7], which guarantees very low latency in service deployment and monitoring.

We demonstrate the capabilities of this platform by demonstrating useful third-party applications, which utilize the

[1] WiFi APs, like all other compute platforms, will continue to get faster and powerful and the capabilities installable in chutes will continue to improve — even our current low-end version can already do some image processing and motion detection in video feeds.

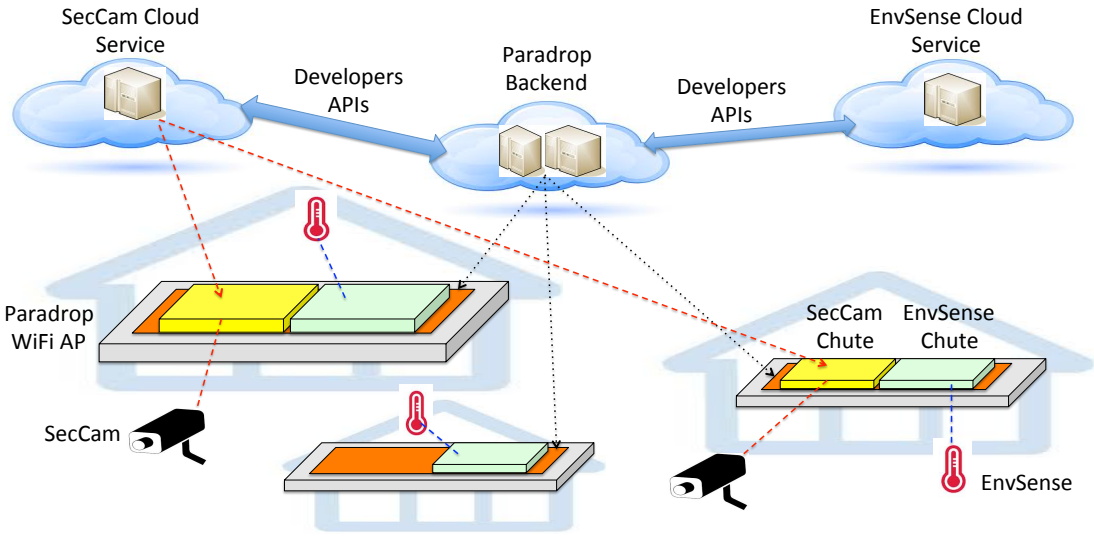

Fig. 1. Overview of Paradrop. There are three components in the system: Paradrop backend, Paradrop gateways, and the developer API. Developers can deploy services in containers (we call them chutes in Paradrop platform, as in parachuting a service on-demand) to the gateways through the backend server. The chutes are created 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 motion; and 2) EnvSense: a system deployed in buildings to monitor temperature and humidity with sensors. We illustrate these two applications in Section V.

Paradrop framework. Figure 1 gives an overview of the architecture of Paradrop platform, and also shows two example applications: a security camera application that connects with in-range cameras for detecting motion, and an environment sensing application that connects with temperature and humidity sensors in homes and interacts with actuators such as thermostats. 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 to some cloud service 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 analyze and evaluate the system in terms efficiency and effectiveness. We also discuss the flexibility and scalability of the system.
- We introduce the approach to develop applications for Paradrop platform with two example applications. And we also discuss other possible applications that can be developed for this platform.

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

II. Paradrop Overview

A. Enabling Multi-tenant wireless gateways and applications through Paradrop

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 computing framework. The gateway – which could be a home WiFi Access Point (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.

But, why would we want to push computation onto the home gateways (e.g., WiFi APs and cable set-top boxes)? We believe there are a few simple reasons:

- The home gateways can handle it. Modern home gateways are much more powerful than what they need to be for their networking workload. Further, if one is not running a web server out of the home, the gateway sits dormant a majority of the time (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, namely many cheap Internet of Things sensor and actuator components. Using Paradrop, developers can offload some computation of one (or even a group of) 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 as controls for home 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 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.

B. 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, 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 through the close proximity footprint of the gateway.

Figure 2 shows a Paradrop gateway, along with two services to motivate our platform: "SecCam" and "EnvSense". The current instance of Paradrop gateways have been implemented on a PCEngines single board computer [8] running Snappy Ubuntu on an AMD APU 1GHz processor with 2GB of RAM. This low-end hardware platform was chosen to showcase Paradrops capabilities with existing gateway hardware. We have ourselves implemented two third-party services by migrating their core functionality to the Paradrop AP platform to demonstrate its potential. 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 [9], 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 "SecCam". Leveraging the Paradrop platform, the two services allow us to motivate the following advantages of Paradrop, if developers design their services right. 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:

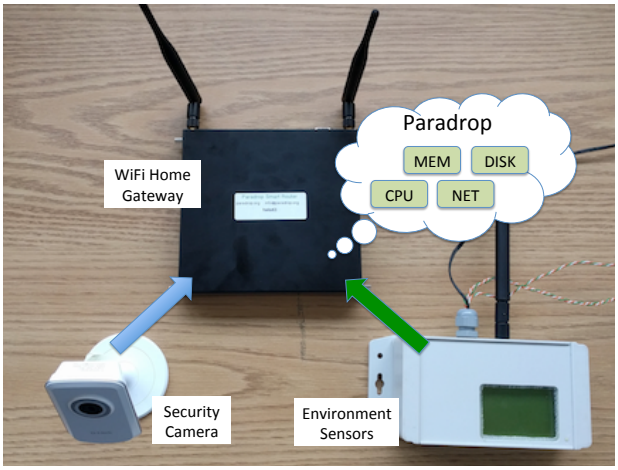

Fig. 2. The fully implemented Paradrop gateway, which shares its resources with two wireless devices including a Security Camera and an Environmental Sensor.

- *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.
- *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 in 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, e.g., proximity of different devices to each other, location in specific rooms, and much more. If such an API is exposed to developers (with care from privacy management), it enables new capabilities that is 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

A. Challenges and Solutions

Virtualization technology is used in Paradrop platform to provide isolated environment to services running on the gateway. One key difference of Paradrop platform compared to cloud computing platform is that the gateway hardware has lower performance than servers in data centers. 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. Container is more lightweight and faster to boot. So we use container-based virtualization rather than VMs to provide isolated environment for the services deployed on the gateways.

Another challenge to design Paradrop platform is that 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 in the platform. To solve this problem, we need to deploy a message router to connect developers' console to the gateways. We use Web Application Messaging Protocol (WAMP) [7] to transmit messages between consoles and gateways.

Unlike servers in data center, after we deploy Paradrop gateways in the network edge - user's home or office, it is harder for us to control the hardware and upgrade software on it. Reliable software update mechanism is important for the project deployment and maintenance. We need a software system with secure and transactional update capability. The software system also needs to support image backup and roll back 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 [10] which meets our requirements.

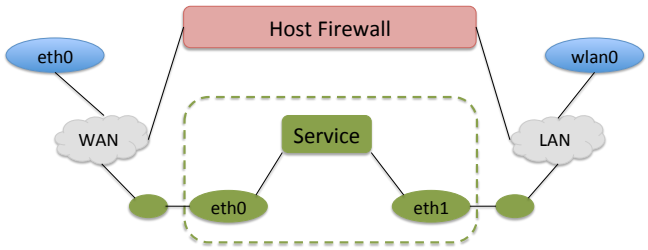

Fig. 3. A chute is running on an AP. The dashed green box shows the block diagram representation of a "chute" installed on a ParaDrop enabled gateway. Each chute hosts a stand-alone service and has its own network subnet.

B. Paradrop System Architecture Overview

Paradrop platform has two core components as shown in figure 1. The backend and gateways are connected by a WAMP message router. The gateways provide virtualized environment for the services. Whereas the backend maintains the information of the gateways and users. It also provides a repository 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

The primary component of Paradrop is the package called a chute (short for parachute) because the framework uses it to install services across different gateways. Each developer can deploy many chutes (figure 3) to their AP, thanks to a low-overhead container technology. These chutes allow for fully isolated use of computational resources on the gateway. As we design and implement services on the gateways; we can, and should, separate these services into unique chutes. Think of each chute as a different application in the Android Play Store (you wouldnt combine a TODO list tracker with a calculator would you?).

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 users 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.

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. The use of SDN is what allows developers to transparently redirect the users request to their web services from within the gateway.

Resource Policies: 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.

IV. Implementation

In this section, we introduce the implementation details of Paradrop platform. Paradrop has evolved from a VM-based version to a Linux Containers (LXC) [4] based architecture, and finaly to the current Docker [6] based architecture, though the design motivations and strategies are kept the same. We only cover the implementation of the latest version in this paper. As we introduce in section III, WAMP is used to connect the Paradrop backend and the gateways. WAMP is an open standard WebSocket [11] subprotocol that provides two application messaing patterns in one unified protocol: Remote Procedure Calls + Publish & Subscribe [7]. These two messaging patterns exactly match our requirements to manage and monitor the Paradrop gateways with minimum latency. We built our system based on Autobahn [12], which provides open-source implementations of the WebSocket Protocol and WAMP. A WAMP router was needed for the Paradrop platform and we selected crossbar.io. Crossbar.io [13] is an open source multi-protocol application router based on Autobahn and WAMP. We installed crossbar.io on a Ubuntu 14.04 machine which is accessible by 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 Paradrop backend with Python programming language based on the Twisted [14] network framework. The major components of the backend are shown in Figure 4. Two important interfaces are implemented for the backend server:

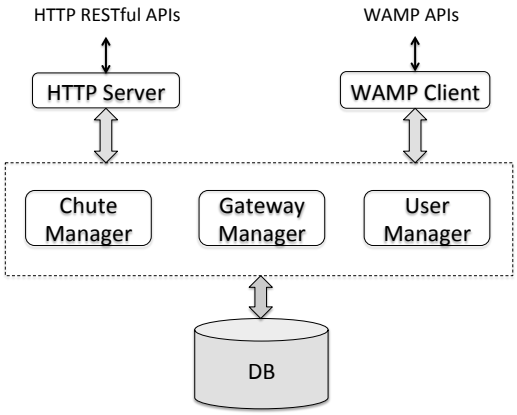

Fig. 4. Architecture of the Paradrop backend. The backend manages all the resources of the Paradrop platform and provides APIs for users to deploy services on the gateways.

- The WAMP APIs for the Paradrop gateways. In the gateway side, the backend communicates with all the gateways to dispatch commands and receive responses and status reports. The soft real-time characteristic of WAMP guarantees the minimal latency in that process [7].
- The HTTP RESTful APIs for users(developers, end-users and administrators). In the user side, the backend aggregates the information from all the gateways, and provides the information to Paradrop frontend, by which the information can be visualized and shown to users. The backend server also stores some persistent information about the Paradrop platform deployment, e.g., the location of the gateways, configuration of the gateways, etc. The backend server also relays the commands from users, e.g., start a chute on one ore more specific gateways, and then relay the responses from the gateways to users.

The backend has authentication mechanisms. All users need to register to the backend first in order to get permissions for the resource. The backend server stores information about the users, gateways and chutes in a MongoDB database [15]. We are in the process to implement a repository to manage the published chutes in the backend server. Users can deploy chutes to their gateways through the web frontend with that repository. Currently the user who wants to deploy a chute to a gateway needs to make sure the chute package is available locally for the Paradrop developer console.

B. The Paradrop Gateway Implementation

Paradrop gateway is the execution engine of the Paradrop platform. First of all, we need to reliably maintain Paradrop software components on the operating system of the gateway. In order to securely and reliably manage the Paradrop software components running on the gateway, we selected Snappy Ubuntu as the operating system. Every software component

is a snappy package in Snappy Ubuntu, and users can transactional upgrade or rollback the component reliably[10]. It is easy to maintain the software on Paradrop gateways in a large deployment with that capability.

On one hand, we want to provide an isolated and virtualized environment for the chutes deployed on the gateway. On the other hand, we want to keep the virtualization overhead minimum because the hardware platform of the gateway is not as powerful as the server in the cloud computing platform. There are two virtualization options for us on the Linux operating system: Virtual Machines (VMs) and containers. While the goal is similar, they virtualize the system in different level. Hypervisor-based VMs virtualize at the hardware layer, while containers virtualize at operating system layer. Felter et al. did measurement study to compare the overhead of VMs and Linux containers. More specifically, they compared KVM [16] with Linux containers [4]. They concluded that KVM's complexity makes it not suitable for workload that is latency-sensitive or have high I/O rates, though both KVM and container have very low overhead on CPU and memory. Moreover since hypervisor-based virtualization provides access to hardware only, we still need to install an operating system in the virtual environment, which quickly gobbles up resources on the gateway, such as RAM, CPU and bandwidth [17]. Therefore we selected Linux containers to build Paradrop platform. Instead of using LXC [5] directly, we built Paradrop based on Docker [6], which is an open-source engine to commoditize LXC. Docker adds features such as layered image and NAT, which ease the development, management and deployment of chutes. Docker also nicely solve the dependency problem when developers migrate software tested in development environment to deployment environment. There are some complexities to build Paradrop with Docker. For example, LXC is an example of OS-containers, whereas Docker is an App-container. Therefore Docker is suitable for the chute that only has one process. If a complicated service needs to run multiple processes, we can either launch multiple containers or use Docker supervisor. Moreover, since Docker is a pure execution environment, we need to manage the networking environment by ourselves. We can not provide a operating system environment including networking configurations in a container like LXC does.

An Paradrop daemon (also called Instance tools) runs on the gateway to implement all the functions related to Paradrop platform. It implements all Docker related features based on a python wrapper of Docker APIs. And it exposes the controlling and monitoring interfaces to outside world. Its major responsibilities include:

- Registers the gateway to the Paradrop backend.
- Monitors gateway's status and report it to the Paradrop backend.
- Receives RPCs and messages from the Paradrop backend, and manage containers on the gateway accordingly, e.g. install, launch, stop, uninstall, etc.

The Paradrop daemon interfaces support both WAMP and HTTP protocols. Therefore Paradrop gateway can be accessed by the developer tools indirectly through the WAMP message router or directly if they are in the same network. That is convenient for developers for issues debugging. And also for the users whose gateways does not have stable access to the backend server.

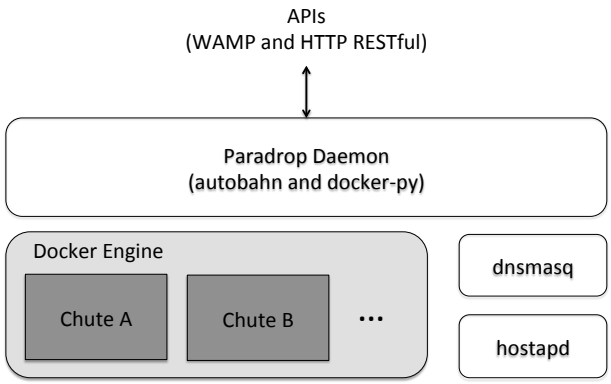

Fig. 5. Major software components on a Paradrop gateway. All the components are snappy packages. We implemented Paradrop daemon with Python from scratch. And we made the snappy packages for dnsmasq and hostapd based on the corresponding open source projects. The Docker engine is packaged by Ubuntu. All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

Paradrop daemon also controls the firewall, DHCP, WiFi, etc. to provide an environment for the chutes managed by Docker engine, and also for the devices connecting to the gateway.

Other than providing run-time environment for the chutes, the daemon also manage the resources on the gateway. Every chute needs to have the parameters to define the resource requirement in a config file. The Paradrop daemon enforces the resource policies when it is creating or starting a chute instance. Currently Paradrop supports policies for below resources:

- 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 when the container is running. The "share" value defines relative share of the CPU resource that one chute can use when it compete with other chutes. It does not limit the maximum CPU resource a chute can use. We will 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.
- Disk size: Currently we only define a common maximum disk size for all chutes. In the future, we may support defining different disk size for different chutes.
- Network: The Paradrop daemon tracks all the network interfaces used by chutes, and restrict their speed by traffic shaping. We might implement a strategy based on "share" value similar to the CPU resource management in the future release.

Some edge computing applications need the storage re-

source to cache content, or to provide local storage services. So 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 Linux kernel's device mapper and Docker's "devicemapper" storage driver [18]. Example policies include disk size quota for a chute, read/write speed, etc.

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 tool is agnostic to the contents of a chute. Essentially a chute is a Docker image along with some Paradrop platform specific files. The developer console provides the tools that we can use to interact with the Paradrop platform. Developers can create chute locally, upload chute to backend, and install chutes from local machine or backend to the gateways that they have permissions to do so. It also provides command line tools to 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 message router.

D. The Paradrop Web Frontend implementation

We are in the process to implement a Web frontend for Paradrop platform. It will provide all the features of the developer console except building a chute. Moreover, it will provide 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 will also be nicer compared to the command line tools.

E. The Pardrop Workflow

In order to clearly show the implementation of the Paradrop platform, this section introduces the workflow that a developer working with the Paradrop platform. Developers need to interact with two components of the Paradrop platform:

- The build tools in developer console - our command line tools that enable registration, login, and control.
- The instance tools - the tools of Paradrop daemon running on the Paradrop gateways to launch chutes on the virtualization substrate.

Figure 6 shows the two components that a developer need to work with. By providing tools to create and manage chutes, the platform implementation is transparent to developers. They can think the Paradrop hardware just some normal resources sit close to user's mobile devices, and focus on the application logic development.

If we want to deploy an application on the Paradrop platform, the first step is to try the application locally, and then we can pack the binary and configuration files into a chute by the build tools. After that we can deploy and debug the chute on a local Paradrop gateway. After we verify the functionality of the chute, we can either install the chutes to routers we want, or publish the chutes to a ChuteStore of the Paradrop platform. Service providers and end-users can install the chute on the specified Paradrop routers. We are in the process to build the ChuteStore. Our goal is to make the chute management easy and flexible for developers, users and administrators.

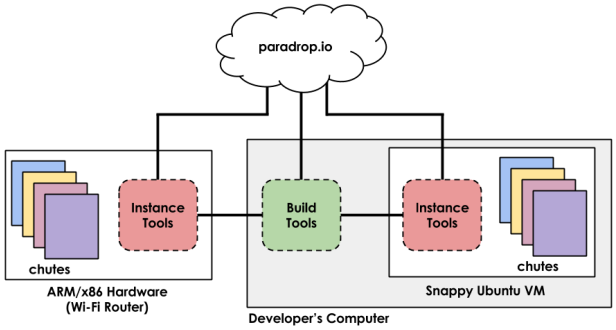

Fig. 6. Paradrop platform from developer's view: We refer to Build Tools when we talk about the command line program running on developer's development computer that control and communicates with the rest of the Paradrop platform. The Instance Tools leverage Docker to allow Paradrop apps to run on router 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).

V. Paradrop Applications

Developers can deploy diverse chutes (applications and services) on Paradrop platform. Any service running on a cloud computing platforms can be easily ported to Paradrop platform with few changes if it can take advantage of edge computing. In order to make the service compatible with Paradrop platform, we need to provide some configuration files to define the resource requirement of service, e.g. CPU, memory, network, etc. With Paradrop developer tools, we can package the binary files, scripts, Dockerfile, and Paradrop configuration files to a chute. Then the chute can be deployed on Paradrop gateways to provide the service. Figure 7 illustrates the process to deploy and launch a chute on a gateway. Depends on the applications requirements, we can either deploy the whole service to the Paradrop platform; or for a complex service composed by many microservices, we can deploy some parts of the service (some microservices) to the Paradrop platform. In the later case, Some microservices running in the cloud will cooperate with the microservices on Paradrop platform to provide the service to end-users.

In this paper, we only present two example applications in details. Both of them are Internet of Things (IoT) applications. IoT is becoming a huge part of the networking 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 back into the gateway.

A. A Security Camera Service Using Paradrop

In this section, we present a walkthrough about using a WiFi-based 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

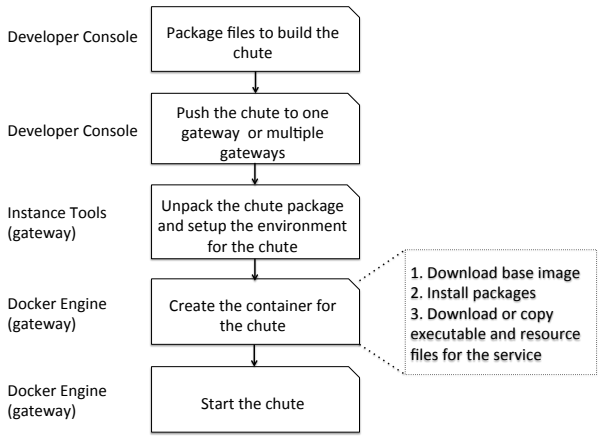

Fig. 7. The process to build, deploy and launch a chute on a gateway. The step to create the container for the chute could be simplified by building the Docker images and uploading them to a private repository.

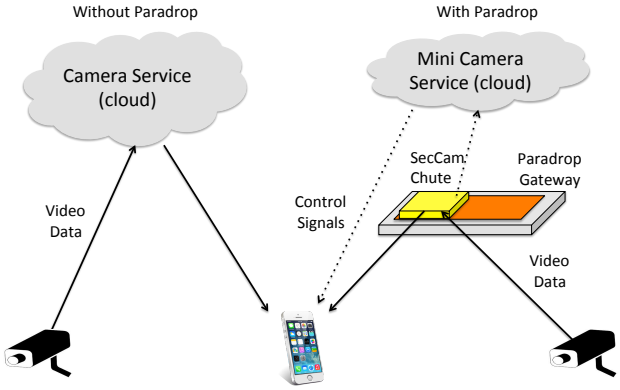

Fig. 8. The IP camera service without Paradrop platform and with Paradrop platform. With Paradrop platform, we can deploy the camera service into the gateway. If the mobile device has direct connection with the camera service on the gateway, it will control the camera and get video data from it directly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

camera (D-Link 931L webcam), where we took the role of developer to fully implement the service.

DLink provides cloud service "mydlink" [19] to which user can register their camera. With that service, users 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 cloud platform and users do not have much control about it. We propose to move the functionality of the cloud service to the Paradrop gateway to give users full control about the camera device and the video data.

For this service, we require networking interfaces to communicate with the webcam and the Internet, as well as ample storage for images. To augment storage resources on Paradrop gateways, we add a flash card to the gateway device which provides GBs of storage. The applications for SecCam allow for motion detection from the webcam, user defined alerts, as well as visualization of the detected images. The motion detection component is a Python based program with user defined characteristics such as threshold of motion, time of day, and rate of detection. Visualization of the motion is implemented as a PHP based web page, which is hosted within the SecCam chute. Figure 8 compares the camera service deployed in cloud and Paradrop platform.

We need to create a chute for the "SecCam" service, which has the following responsibilities:

- **Create the SecCam SSID.** This SSID provides an isolated WiFi network and subnet to the security cameras. This is designed so that devices purchased by end-users do not have to be programmed when they arrive at the house (they can be flashed with a default SSID and password by the company). This subnet will not have internet access, and any network traffic be consumed by the chute.
- **Image capture service.** The service will run a simple Python program to capture images from an IP camera, calculate differences to detect motion, and store those images to disk. The images stored to disk will then be visualized using a web server which runs inside the chute.
- **Web server.** The web server provide service to users in local network to view the video and check the logs. We implemented a very simple web server with PHP.

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

- The Dockerfile follows the specification of Docker. It defines the Docker image which will be managed by the Docker engine on the Paradrop gateways.
- The Paradrop configuration file is a YAML formatted file for Paradrop platform. It describes the resource requirement of the chute.

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 are installed on the image when we launch the chute on the Paradrop gateway. Alternatively we can store the pre-built image in a private repository so that we can launch the chute directly.

B. The EnvSense Service

This service is a wireless environmental sensor designed as part of the Emonix research platform [9]. Since the service is fully implemented, we only need to migrate the service, rather than rewrite it to fit Paradrop platform. The original service runs in a server to collect data from the sensors, process and store the data, and visualized the data. After identify the resources required to run the service, we can create a configuration file and then create a chute. The steps to create

```
owner: Paradropdate: 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
      dhcp:
         lease: 12h
         start: 100
         limit: 50
resource:
   cpu: 1024
   memory: 128M
   wan:
      down: 25000
      up: 10000
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.

the configuration file are similar to the SecCam service so we omit them here for brevity.

As a future work, we will divide the EnvSense service to be multiple microservices. E.g. we want to run data collection and processing microservices on the Paradrop platform, and run data storage and visualization services on the cloud platform. Then we need to modify the original implementation, however it will be fairly easy with the development and management tools provided by Paradrop.

C. Other possible applications

Other than IoT, applications in other categories can also take advantage of the Paradrop platform. For instance, P2P technology is a well known approach to reduce the workload of media server. However, it is challenging to use P2P on mobile devices because of below issues:

- Mobile devices are powered by battery, so that it 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 wired network, wireless network is dynamic, so that the overhead to maintain the status of peers can be too high and offset the potential advantages of P2P technology.

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

We also built a media caching service on the Paradrop platform to prefetch and cache video data from the media servers, e.g. Netflix. That service improves user experience by eliminating network congestions. Some applications that use edge computing to optimize the bandwidth usage of wireless applicantions have been developed by other researchers. For example, Zhang et al. use computing resource in network edge to optimize the wireless video surveillance system [20]. Such kind of applications can be easily deployed in Paradrop platform if the application can run on Linux operating system.

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%

VI. SYSTEM EVALUATION

A. Efficiency of virtualization

Virtualization solution is the core of Paradrop platform. We implemented three generations of virtualization scheme in the Paradrop platform evolution.

- **Hypervisor-based virtualization: OpenWRT [21] + lguest.** In the first generation of Paradrop platform, we implemented the virtualization solution based on lguest. Lguest is a small x86 32-bit Linux hypervisor for running Linux under Linux [22]. A number of hypervisor solutions have appeared that use Linux as the core, including KVM [16] and lguest. We selected lguest because of its relatively simple implementation (5000 lines of code) and its availability in the mainline Linux kernel to our first generation hardware platform. Though the simplicity of lguest is attractive, it is slower than other hypervisors. And recent measurement study by Felter et al. confirmed that VMs have high latency in I/O operations [17] and they are not suitable to latency sensitive applications. Another problem of lguest is that it is a paravirtualization hypervisor, which means the guest OSes need to be lguest-enabled. That restriction increases the efforts to port a service to the Paradrop platform.
- **Linux Container: OpenWRT + LXC.** We implemented the second virtualization scheme with Linux container (LXC), which is more lightweight than hypervisor-based virtualizations. Containers have low overhead than hypervisor-based virtualization scheme because they do not emulate hardware, and they share the same operating

system as the host. There are two user-space implementations of containers, each exploiting the same kernel features [5]. We selected "LXC" because it is flexible and has more userspace tools. In order to implement container-based virtualization scheme, the virtualization 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 [23] and the results are shown in Table I. 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. Lguest needs much longer time because we needed to boot the guest operating system before we can run an application. Packet latency results also show the advantage of LXC. Table I also shows the throughput fairness, LXC can achieve good fairness along with much lower overhead than lguest.

- **Linux Container: Snappy Ubuntu + Docker** Though the second generation virtualization scheme have very low overhead, and the implementation on OpenWRT is very efficient for the hardware platform we selected for the first generation Paradrop gateway. We began to realize that the operating system disparity on the gateway and the cloud platform could be an obstacle to develop or port applications to Paradrop platform. Therefore we switched to Snappy Ubuntu while we upgraded the hardware platform to a more powerful one with 64bit processor. We also made a big change on the software architecture by switching from LXC to Docker. Although Docker and LXC share the same underlying container-based virtualization technology, Docker provides much easier to use tools for us to implement the Paradrop platform. And they have similar efficiency. To have a clear idea about the latency to deploy, start, stop, and destroy a chute on a gateway, we measured the time taken for these operations. The results are shown in Table II. The hardware platform is: AMD G Series T40E APU, 1GHz Dual Core, 2GB DDR3 DRAM. We tested the system with the chute of SecCam application we described in sectionV. The test results depend on the network bandwidth because we need to download the Ubuntu 14.04 based image from the Docker repository, some Ubuntu packages from an Ubuntu repository, and a package including the files for the web pages along with the Python implementation of the service from a file server. The steps to install packages on the base image are also time consuming. Figure 11 illustrates the time taken by each step.

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

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

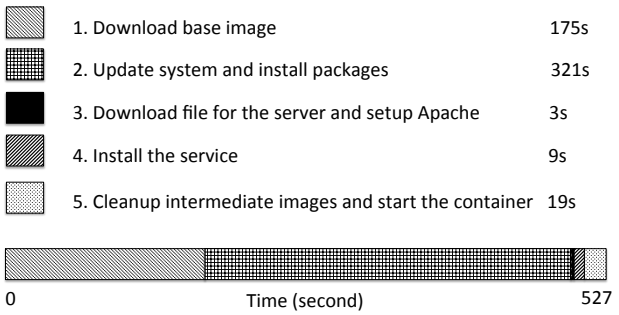

Fig. 11. Time taken by each step in a chute deployment. Step 2 took 321 second, which is more than 50% of the total time. If an application is sensitive to deploying time, we can pre-build and store the image for that application in the private repository, then step 2 can be eliminated.

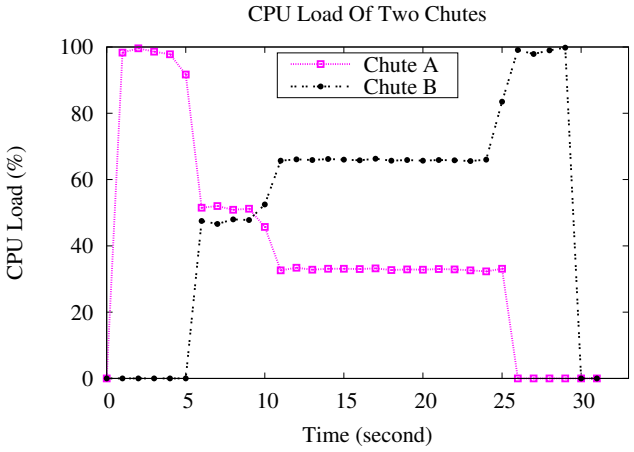

Fig. 12. CPU resource management test. We started two chutes from the beginning. Chute A and B have share values 512 and 1024 respectively. Both of them can stress test the CPU when they work. Chute A started working from the beginning, and became idle after 25 seconds. Chute B started working 5 seconds 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 second, 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 Paradrop platform, we developed two test chutes and deployed them on a gateway to stress test the system. Figure 12 shows the result. Docker uses cgroups to group processes running in a container. This allows us to manage the resources for containers. When we build a container, we can specify a 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

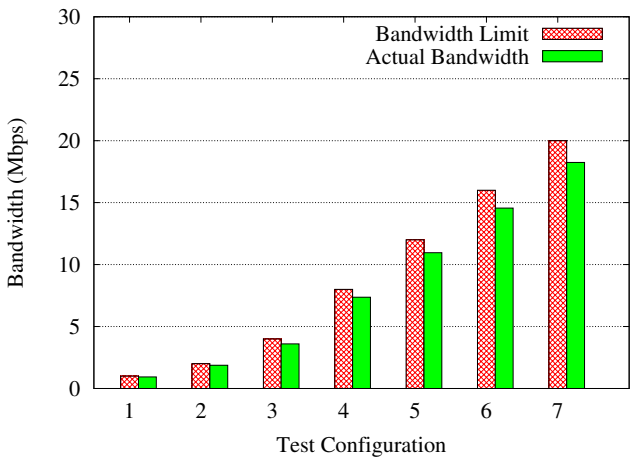

Fig. 13. Network speed limit test. We launched a test chute including a HTTP server and a 100MB file. In order to avoid the network bandwidth variation of wireless interfaces, we conducted the test with an Ethernet interface. We limited the network bandwidth of the chute to be seven different values,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

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. It is fairly easy to extend the system to add that support if necessary.

By default every chute can use all the memory resource 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 needs to note is that we can also control the swap available for a chute. By default we disable the swap usage 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 allocate 256MB memory in start. 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.

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

Paradrop gateway has an SD card with 16GB capacity. 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, 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 carefully control the speed that a chute to access the file system or else one chute would slow down all other chutes easily. We plan to add the capability to control the I/O speed in the future.

C. Flexibility and convenience to deploy services in the edge

Paradrop platform is highly flexible to deploy edge computing applications. The flexibility is guaranteed by below design choices.

- The operating system on the Paradrop gateway is Snappy Ubuntu. It provides secure and transactional update capability to us to reliably manage the software on the gateways. Moreover, Snappy Ubuntu is similar to the OS in the cloud computing platform, so developer can port or develop applications for Paradrop easily.
- 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, developer can use different versions of Python to develop the services. That flexibility makes Paradrop platform easy to be accepted by developers. Moreover, 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 cloud computing platform to Paradrop platform is very low. The only requirement of Paradrop platform on a service is that the service should be able to run on relatively new Linux kernel. We believe that will not be a problem in practice. Compared to develop an application for Android or iOS, developers of Paradrop platform do not need to install the SDKs or learn new application frameworks.
- WAMP based messaging mechanism simplified the deployment of Paradrop gateways. Paradrop platform leverages both the remote procedure call and publish/subscribe messaging schemes to implement the communications between Paradrop backend server and Paradrop gateways. Developers do not need to maintain direct connections to the gateways in order to debug their chutes in deployment.

D. Scalability

By splitting and distributing the services on the cloud to the Paradrop gateways, Paradrop platform provides a good platform to deploy services in large scale. In the future, we plan to only transmit latency-sensitive messages with WAMP,

and transmit other messages with HTTP protocol. Then the WAMP message router (crossbar.io) will not be the bottleneck of the system even with a large number of gateways. 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. Academia and industry have explored different approaches to virtualize 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 fast evolution of virtualization technologies. Virtualization is widely used to improve the resource utilization, and to simplify the data center management. Hypervisor-based virtualizations, e.g. Xen [24], VMware ESX [25], KVM [16], lguest [22], etc. emulate hardware resources to the guest operating system and they can achieve high flexibility. But 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 advantage on efficiency over hypervisor-based virtualization. Examples include FreeBSD jails [26], Solaris Containers [27], etc. Because of the availability of supporting technique in Linux kernel like namespace [28], cgroups, etc. Container-based virtualization is becoming a popular virtualization approach for the Linux operating system [29], [4]. In addition to the applications on cloud computing in data centers, researchers predicated the virtualization will be widely used in other environments, e.g. desktops, smartphones, etc. [30]. Paradrop leverages the virtualization technology to provide isolated, controlled environment to the services deployed in the network edge. We are using it in a distributed, resource restricted environment. Whereas cloud computing uses virtualization technology in centralized located and managed data centers with high performance hardware.

CPU offloading. CPU offloading approaches propose to offload computationally intensive tasks from the battery driven mobile devices to co-located specialized devices or cloud in order to improve the battery life of mobile devices or performance. MAUI [31] and ThinkAir [32] 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 the edge computing and proposed different approaches to leverage 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 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, etc. make it the appropriate platform for a number of 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 cloud and distribute compute load among cloud servers (cloudlet) to minimize the response time [33]. 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 [20].

Smart routers. As the performance of WiFi routers keep increasing, many companies are interested to build smart routers that can be managed and monitored by mobile applications [34], [35]. Users can even install third-party applications on some of them [36]. Their goal is to optimize the user experience of mobile devices users, whereas Paradrop has a different goal to push services from data centers to the network edge.

VIII. Conclusion and Future Work

Paradrop platform 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, and 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 at 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), audio sensors, etc. to further improve the capabilities of the gateways. 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.

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

[6] (2016) Docker homepage. [Online]. Available: https://www.docker.com/

[7] (2016) Wamp: The Web Application Messaging Protocol. [Online]. Available: http://wamp-proto.org/

[8] (2016) PC Engines apu platform. [Online]. Available: http://www.pcengines.ch/apu.htm

[9] 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.

[10] (2016) Snappy Ubuntu. [Online]. Available: https://developer.ubuntu.com/en/snappy/

[11] I. Fette and A. Melnikov, "The websocket protocol," RFC 6455, 2011.

[12] (2016) Autobahn: Open-source real-time framework for Web, Mobile and Internet of Things. [Online]. Available: http://autobahn.ws/

[13] (2016) Crossbar.io: Connecting Systems, Devices and People. [Online]. Available: http://crossbar.io/

[14] (2016) Twisted matrix labs: Building the engine of your Internet. [Online]. Available: https://twistedmatrix.com/trac/

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

[16] 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.

[17] 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.

[18] (2016) Docker and the Device Mapper storage driver. [Online]. Available: https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/

[19] (2016) mydlink. [Online]. Available: https://www.mydlink.com/entrance

[20] 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.

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

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

[23] 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.

[24] 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.

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

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

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

[28] 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.

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

[34] (2016) Smart wifi app center. [Online]. Available: http://www.linksys.com/us/smart_wifi_center

[35] (2016) Meet OnHub. A new type of router for the new way to Wi-Fi. [Online]. Available: https://on.google.com/hub/

[36] (2016) HiWiFi Apps. [Online]. Available: http://www.hiwifi.com/j3-func