1 ParaDrop: An Edge Computing Platform in Home Gateways

SUMAN BANERJEE,[1] PENG LIU,[1,2] ASHISH PATRO,[1] and DALE WILLIS[1]

[1]*Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, USA*
[2]*Pennsylvania State University, State College, PA, USA*

1.1 INTRODUCTION

The last decade has seen a rapid diversification of computing platforms, devices, and services. For example, desktops used to be the primary computing platform until the turn of the century. Since then, laptops and more recently handheld devices such as laptops and tablets have been widely adopted. Wearable devices and the Internet of things (IoT) are the latest trends in this space. This has also led to widespread adoption of the "cloud" as a ubiquitous platform for supporting applications and services across these different devices.

Simultaneously, cloud computing platforms, such as Amazon EC2 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 require developers to host services, applications, and data on off-site data centers. But, due to application-specific reasons, a growing number of high quality services restrict computational tasks to be colocated with the end user. For example, latency-sensitive applications require the backend service to be located to a user's current location. Over the years, a number of research threads have proposed that a better end-user experience is possible if the computation is performed close to the end user. This is typically referred to as "edge computing" and comes in various flavors including: cyber foraging [1], cloudlets [2], and more recently fog computing [3].

Fog for 5G and IoT, First Edition. Edited by Mung Chiang, Bharath Balasubramanian, and Flavio Bonomi.
© 2017 John Wiley & Sons, Inc. Published 2017 by John Wiley & Sons, Inc.

This chapter presents a unique edge computing framework, called ParaDrop, which allows developers to leverage one of the last bastions of persistent computing resources in the end customer premises: the gateway (e.g., the Wi-Fi access point (AP) or home set-top box). Using this platform, which has been fully implemented on commodity gateways, developers can design virtually isolated compute containers to provide a persistent computational presence in the proximity of the end user. The compute containers retain user state and also move with the users as the latter changes their points of attachment. We demonstrate the capabilities of this platform by demonstrating useful third-party applications, which utilize the ParaDrop framework. The ParaDrop framework also allows for multitenancy through virtualization, dynamic installation through the developer API, and tight resource control through a managed policy design.

1.1.1 Enabling Multitenant 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 smartphones 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 that provides all the capabilities developers require for their services, as well as the proximity expected from an edge computational framework. The gateway—which could be a home Wi-Fi 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., Wi-Fi APs and cable set-top boxes) for the 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. What is more if you are not running a Web server out of the house, your gateway sits dormant majority of the time (when no one is home using it).
- Utilizing computational resources in the home gateway gives us a footprint within the home to devices that are starved for computational resources, namely, IoT devices. Using ParaDrop, developers can piggyback their IoT devices onto the AP without the need for cloud services OR a dedicated desktop!
- Every household connected to the Internet by definition must contain an Internet gateway somewhere in the house. With these devices sitting around, we can use them to their full potential.

- Pervasive Hardware: Our world is quickly moving toward 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. In this chapter, we examine the requirements of services in order to build an edge computing platform, which enables developers to provide services to the end user in place of a cloud computing platform. A focus on edge computation would require developers to think differently about their application development process; however we believe there are many benefits to a distributed platform such as ParaDrop. The developer has remained our focus in the design and implementation of our platform. Thus, we have implemented ParaDrop to include a fully featured API for development, with a focus on a centrally managed framework. Through virtualization, ParaDrop enables each developer access to resources in a way as to completely isolate all services on the gateway. A tightly controlled resource policy has been developed, which allows fair performance between all services.

1.1.2 ParaDrop Capabilities

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

Figure 1.1 shows ParaDrop system running on real hardware, the "Wi-Fi home gateway," along with two services to motivate our platform: "security camera" and "environment sensors." ParaDrop has been implemented on PC engines ALIX 2D2 single board computer running OpenWrt "Barrier Breaker" on an AMD Geode 500 MHz processor with 256 MB of RAM. This low-end hardware platform was chosen to showcase ParaDrop's capabilities with existing gateway hardware.

We have emulated two third-party developers who have migrated their services to the ParaDrop platform to showcase the potential of ParaDrop. 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 environmental sensor designed as part of the Emonix research platform [4], which we refer to as "EnvSense." The second service is a wireless security camera based on a commercially available D-Link DCS 931L webcam, which we call "SecCam." Leveraging the ParaDrop platform, the two developer services allow us to motivate the following characteristics of ParaDrop:

- *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.

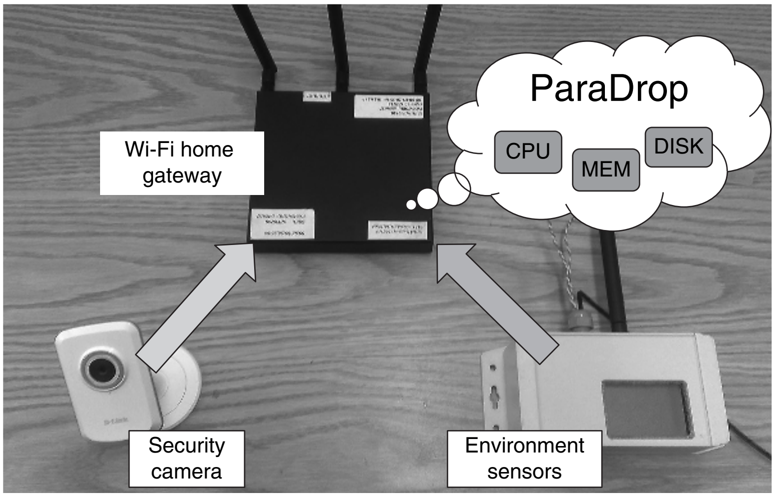

Figure 1.1 The fully implemented ParaDrop platform on the Wi-Fi home gateway, which shares its resources with two wireless devices including a security camera and environment sensor.

- *Low Latency.* Many simple processing tasks required by sensors are performed in the cloud today. By moving these simple processing tasks onto gateway hardware, one hop away from the sensor itself, a reliable low-latency service can be implemented by the developer.
- *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 Context.* In the typical service implemented by a developer, the data is consumed only by the end user yet stored in the cloud. This requires data generated by a security camera in the home to travel out to a server somewhere in the Internet and upon the end user's request travel back from this server into the end-user device for viewing. Utilizing the ParaDrop platform, a developer can ensure that only data requested by the end user is transmitted through Internet paths to the end-user device.
- *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.

1.2 IMPLEMENTING SERVICES FOR THE PARADROP PLATFORM

The primary component of ParaDrop is the virtual machine called a chute (short for parachute) because the framework uses it to install services across different APs. Each developer can deploy many chutes (Figure 1.2) to their AP, thanks to a low-overhead virtualization technology: Linux containers (LXC). These chutes allow for fully isolated use of computational resources on the AP. As you design and implement services on your AP, you can, and should, separate these services into unique chutes. Figure 1.3 shows an example chute configuration specified in the *Chute.struct* file.

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

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

ParaDrop API. The focus of ParaDrop is to enable third-party developers to provide high quality services to their users. In order to enable this, a seamless API was developed, based on a RESTful paradigm, which allows the developer to have complete control over the configuration of their chutes.

Developers can use the API to query and monitor the status of the Paradrop platform:

- Persistent State: Users (type, permissions, etc.), chutes (description, resource requirements, etc.), and gateways (configuration, accessories, location, etc.)
- Real-Time State: Running status of chutes and gateway

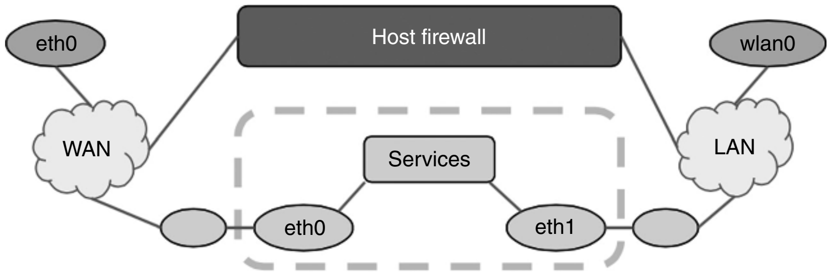

Figure 1.2 The dashed box shows the block diagram representation of a "chute" installed on a ParaDrop-enabled access point. Each chute hosts a stand-alone service and has its own network subnet.

```
"disk": {
  "size": 123456
},
"net": {
  "wan": {
  "type": "wan",
  "intfName": "eth0",
  "ipaddr": "10.100.10.1",
  "netmask": "255.255.255.0"
},
"wifi": {
  "type": "wifi",
  "intfName": "eth1",
  "ipaddr": "10.100.11.1",
  "netmask": "255.255.255.0",
  "ssid": "Virtual0",
  "encryption": "psk2",
  "key": "wifi1234"
}
```

Figure 1.3 An example Chute.struct file, which is used to specify the key configuration parameters of a chute that hosts a stand-alone service. Parameters such as CPU, memory, disk requirements, and network configurations are specified as JSON key–value pairs. ParaDrop provides chute configuration templates to developers, which can customized based on application requirements.

Developers can also use the API to control the system:

- Publish a chute to the store or remove a chute from the store.
- Register/unregister a gateway.
- Install, start, and revoke a chute on one or many gateways.

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 back end, 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 is 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 user's request to their Web services from within the gateway.

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

1.3 DEVELOP SERVICES FOR PARADROP

IoT is becoming a huge part of the networking world. Yet many IoT devices rely on back end services that must traverse the Internet to utilize their full potential. Using ParaDrop, we can pull that intelligence back into the AP.

1.3.1 A Security Camera Service Using ParaDrop

In this section, we present a walk-through about using a Wi-Fi-based video camera with a ParaDrop AP to implement a security camera service called SecCam.

The SecCam service is based on a commercially available wireless IP camera, where we took the role of developer to fully implement the service.

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, and 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.

This example in the section creates a chute for the “SecCam” service with the following end result:

- *Create the SecCam SSID.* This SSID provides an isolated Wi-Fi 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.

```
"disk": {
  "size": 123456
},
"net": {
  "wan": {
  "type": "wan",
  "intfName": "eth0",
  "ipaddr": "10.100.10.1",
  "netmask": "255.255.255.0"
},
"wifi": {
  "type": "wifi",
  "intfName": "eth1",
  "ipaddr": "10.100.11.1",
  "netmask": "255.255.255.0",
  "ssid": "SecCam",
  "encryption": "psk2",
  "key": "noOneCanHackThis"
}
```

Figure 1.4 The primary Chute.struct component for the SecCam chute.

```
{
  "name":"www",
  "path":"/srv/www",
  "location":"@paradrop.server(seccam/srv.tar.gz)",
  "sha1":"526bb8cb52458aad4043c56980cd238551b46b7e",
  "todo":"EXTRACT"
}, {
  "name":"root",
  "path":"/root",
  "sha1":"1633ea1d6351929cc2c8717d1611dcb41681b585",
  "location":"@paradrop.server(seccam/seccam.py)"
}
```

Figure 1.5 The Chute.files component lists the files required for the SecCam chute.

1.3.1.1 Defining the SecCam Chute *Chute.struct.* As discussed earlier, we first need to define the primary *Chute.struct* component first for our awesome SecCam chute (Figure 1.4).

Chute.files. For a chute, the *Chute.files* component lists any files that must exist on the chute's disk in order for it to operate properly. This can include things like bash scripts, Python programs, PHP code, etc.

The rules in Figure 1.5 show files required for our SecCam application. The "www" attribute specifies Web server PHP code to download *seccam/srv.tar.gz* from an *examples* directory on the ParaDrop server to the chute's root file system (FS). Similarly the "root" attribute downloads seccam.py to /root. The "sha1" values let the code running on ParaDrop to verify it properly downloading the code into the chute before it launches.

Chute.resource. As much as possible, ParaDrop tries to be a lean virtualized platform (hence our use of LXC over more traditional virtualization methods). For this reason, we explicitly make the developer define and be aware of the resources they will require for their chute.

These resources are broken down into three categories:

1. *CPU.* The CPU shares devoted to this chute, in most cases the default value, will be fine; if you know the chute will not perform CPU intensive tasks or you want to lower the priority of the tasks it will perform, you can lower the CPU value, by default it is 1024 (meaning equal sharing between all chutes).
2. *Memory.* The AP we have implemented for ParaDrop contains 2 GB of DDR3 memory, so compared with a typical AP memory will not be hard to come by. The default value for memory should typically be fine, but keep in mind: the memory value is a hard limit; if you define it to be too low, your chute's kernel may not even fully boot due to out-of-memory (OOM) issues.
3. *Networking.* The final resource to be defined for chutes is any network throughput requirements of the chute. These are specified in kbps for both upload and download for each interface in the chute. If you are designing a chute with low priority but its use is primarily a virtual router, rather than lowering the CPU resources (which will not greatly affect throughput rates), you should lower the overall throughput provided to the interface instead.

Figure 1.6 shows the *Chute.resource* component for the SecCam chute. We choose the default CPU and memory configuration and specify a high-bandwidth limit to allow high-volume video traffic from the Wi-Fi camera.

Chute.runtime. The *Chute.runtime* component specifies what operations will be performed within the chute itself. We refer to these as the runtime rules (Figure 1.7). The webhosting runtime attribute creates an instance of uhttpd with the arguments specified. The DHCP server runtime macro sets up a default DHCP server inside the chute so that future security cameras can connect to it properly.

Chute.traffic. In many situations, the chute you are implementing will need to interface with devices that for any number of reasons may not be associated to your

```
"cpu": "@resource.cpu.DEFAULT",
"memory": '@resource.memory.DEFAULT',
"wan": {"down": 25000, "up": 10000},
"wifi": {"down": 25000, "up": 10000}
```

Figure 1.6 The Chute.resource component specifies the resource consumption limits for the SecCam chute.

```
{
  "name": "webhosting",
  "program": "uhttpd",
  "args": "-p 80 -i .php=/usr/bin/php-cgi -h /srv/www"
}, {
  "name": "DHCP Server",
  "program": "@net.runtime.dhcpserver"
}
```

Figure 1.7 The Chute.runtime component for the SecCam chute.

```
{
  "name": "Web",
  "description": "Allows the chute to provide a webserver
                  on WAN",
  "rule": "@net.traffic.redirect(@net.host.lan:*:5000,
           wifi:10.100.13.1:80)"
}, {
  "name": "HostSSH",
  "description": "Allows the host stack access to SSH",
  "rule": "@net.traffic.redirect(@net.host.lan:*:5001,
           wifi:10.100.13.1:22)"
}
```

Figure 1.8 The Chute.traffic component allows users to access data within the SecCam chute.

chute's network directly (via a Wi-Fi interface). In these cases for security purposes, the ParaDrop platform allows the developer to implement traffic rules. These rules are implemented in the host networking stack's firewall rules and allow for things like a computer on the host LAN network to access a particular port within a deployed chute (called port forwarding in firewall land).

For the SecCam application, the images are stored within the chute but need to be accessible to users on the LAN network. The Web rule allows the user connected on the LAN network to access Web pages hosted by a uhttpd Web server running inside a chute. The host SSH rule allows the user to SSH into the chute from his laptop (mainly for debugging) connected to the LAN network by using the default ParaDrop SSID (Figure 1.8).

1.3.2 An Environmental Sensor Service Using ParaDrop

Since the wireless environmental sensor was fully implemented as a part of the Emonix research platform, we only need to migrate the service, rather than rewrite it to fit ParaDrop platform. The original service runs in a cloud server to collect data from the sensors, process and store the data, and visualize the data. After identifying the resources required to run the service, we can develop a chute for it so that the service can run in ParaDrop gateways, which are close to the sensors. As the steps to develop a chute for it are the same as the SecCam application, we do not discuss them in detail here.

REFERENCES

1. R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang. The case for cyber foraging. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, pages 87–92, New York, NY, USA, 2002. ACM.
2. M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, 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*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.
4. N. Klingensmith, D. Willis, and S. Banerjee. A distributed energy monitoring and analytics platform and its use cases. In *Proceedings of the Fifth ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, BuildSys'13, pages 5:1–5:8, New York, NY, USA, 2013. ACM.