Cloudrone: Micro Clouds in the Sky

Arjuna Sathiaseelan
University of Cambridge
as2330@cam.ac.uk

Adisorn Lertsinsrubtave
University of Cambridge
al773@cam.ac.uk

Adarsh Jagan
National Institute of Technology, Trichy

Prakash Baskaran
National Institute of Technology, Trichy

Jon Crowcroft
University of Cambridge
jac22@cam.ac.uk

ABSTRACT

This paper presents *Cloudrone*- a preliminary idea of deploying a lightweight micro cloud infrastructure in the sky using indigenously built low cost drones, single board computers and lightweight Operating System virtualization technologies. Our paper lays out the preliminary ideas on such a system that can be instantaneously deployed on demand. We describe an initial design of *Cloudrone* and provide a preliminary evaluation of the proposed system mainly focussed on the scalability issues of supporting multiple services and users.

1. INTRODUCTION

Providing access to the Internet has no one size fits all solution for enabling wider universal access but it requires exploring a variety of solutions. This is evident from organizations like Facebook and Google who are on a mission to connect the next three billion through novel ways by utilizing high altitude platforms such as drones, balloons and satellites.

Access to services are also extremely important (even potentially life saving) during humanitarian crisis and disaster scenarios where access to services are challenged - due to intermittent connectivity, heavy interference and congestion leading to increased latencies to global servers or in most cases no access to them. Hence its not rocket science to understand that access to localized service infrastructures is of paramount importance to solve the global access problem.

Recent work on mobile edge computing [9] aims to push computation right at the edge of mobile networks, enabling computations at the edge improving latencies and performance. The recent work on infrastructure mobility [3] illustrates the interesting concept of making the access infrastructure mobile thus providing better and much more efficient coverage based on the need/demand from the users. So an obvious question is why cannot we directly provision the cloud services right nearer to the user on demand even in the absence of a terrestrial infrastructure e.g. rural/remote areas or disaster zones?

In this paper, we present *Cloudrone*- a preliminary idea of deploying an adhoc lightweight micro cloud infrastructure in the sky using indigenously built low cost drones, single board computers and lightweight Operating System (OS) virtualization technologies such as unikernels/dockers [6]. Our paper lays out the preliminary ideas on such a system that can be instantaneously deployed on demand.

DroNet'16, June 26 2016, Singapore, Singapore
© 2016 ACM. ISBN 978-1-4503-4405-0/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2935620.2935625

2. CLOUDRONE DESIGN

To enable an adhoc lightweight micro cloud infrastructure in the sky using drones, we bring together a couple of fascinating recent innovations in computing: single board computers such as the Raspberry PIs (PI) and lightweight OS virtualisation technologies such as Dockers[1] and integrate them with lightweight quadcopters. There were two possibilities for us to build the *Cloudrone*: integrate the PI with an off the shelf ready made quadcopter or design and build an entire quadcopter from scratch with the PI powering both the drone as well as acting as a micro cloud server. We decide to go for the latter since we wanted to have the drone as lightweight and low cost as possible. We used the PI 2 model B equipped with quad-core ARMv7 CPU 900 MHz and 1024 MB RAM memory. It requires power feed about 800 mA powered using a Lithium Polymer battery (4200mAh 3S 35C) . The PI runs an ARM version of Ubuntu 14.04. We used a Edimax WiFi dongle (EW-7811Un) for providing the WiFi interface. The design and tuning aspects of our quadcopter can be found on the longer version of this paper [7].

The quadcopter's final weight is approximately 1.5 kgs. The maximum altitude to which it was flown was 50 feet while flight time is about 15-20 minutes. The quadcopter's CPU consumption was also measured during the flight operation by using *sysstat* tools. On average our quadcopter consumes 30% for the flight control of the quadcopter leaving 70% for other processes.

2.1 Integrating the micro cloud with the drone

Our vision to build an adhoc micro cloud infrastructure in the air is to use a swarm of PIs on drones acting as adhoc micro cloud servers. The lightweight nature of docker containers significantly reduce the size of image compared to the heavyweight virtual machines (VMs) [5]. For instance,

[1]www.docker.com

we can build a minimal static web server with only 2MB size.

To interconnect the PIs as a swarm/mesh, each of these devices also act as mobile routers operating in two WiFi communication modes: one operates in the WiFi ad-hoc mode to allow Optimised Link State Routing (OLSR) protocol, constructing a Mobile Ad hoc Network (MANET) [2]; the other operates in the Access Point (AP) mode to allow user devices to connect with DHCP. The benefit of using OLSR protocol compared to other MANET routing protocols is it uses a special mechanism called Multi-Point Relay (MPR) to reduce the number of flooded messages. Only a few devices that are located in strategically better spatial positions are chosen to relay (i.e., re-transmit) messages in the path from a source device to a destination device. The MPR mechanism helps reduce overall energy consumption.

We use the Docker swarm technology to create a cluster of multiple docker hosts and migrate the service containers across the cluster. Specifically, there are two types of nodes classified in the swarm. 1. Swarm manager - which manages the overall resources (e.g., swarm members, number of running containers) and decides where to place the service containers. 2. Swarm agent - nodes registered with the swarm manager. When the swarm manager and agents are created, they have to register with the discovery backend as members of the swarm. The discovery backend maintains an up-to-date list of swarm members with the swarm manager. The swarm manager uses this list to assign tasks and schedule the service containers to the agents. To determine where to place the new container in the swarm, the swarm manager uses either *spread* or *bin packing* strategy to compute the rank regarding node's available CPU, RAM and the number of running containers. With the *spread* strategy, the swarm manager gives a priority to the node who has the largest available memory or has the minimum number of running containers. On the other hand, the *bin packing* strategy tries to pack as many containers to a node until reaching its maximum capacity (e.g., RAM, CPU).

2.2 Deploying the Cloudrone

Cloudrone targets to provide localised adhoc service infrastructure in challenged network environments. Such scenarios refer to the post-disaster situations and humanitarian crisis wherein traditional communication services are completely inoperable. An example of *Cloudrone*'s deployment is illustrated in Figure 1. A base camp command center with backhaul Internet connectivity (e.g., satellite link) is set up close to the target area. A cluster of drones can form a mesh network and provide localised services to the users on the ground via WiFi. A variety of (crucial) services (lightweight docker containers) can be either pre-loaded onto the PI or on demand from the ground. The MANET of drones, facilitates the swarm manager to communicate and control the cluster remotely from the base camp through the long haul link. The swarm manager can update the necessary services from the Internet and disseminate throughout the cluster.

In some operations, the cluster can be out of contact with the swarm manager (i.e., the target area is far away from the command center). To deal with any interruption of service, we can create a primary swarm manager operating as the main point of contact and multiple replicas to be the backup swarm managers. Using this feature, the replicas can seamlessly take over the functionalities from the primary swarm manager when it fails. If the cluster fails to contact the primary swarm manager, the most powerful replica automatically takes over the control.

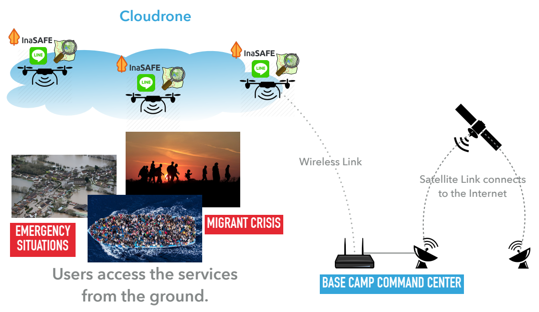


Figure 1: An example of Cloudrone's operation

3. PRELIMINARY BENCHMARKING

The preliminary benchmarking presented in this paper focusses on understanding the scalability issues of lightweight OS virtualisation technology such as Docker on a PI. We focus on two main questions 1) how many Docker containers can a single PI support? 2) how many user requests can a single container running on a PI support?

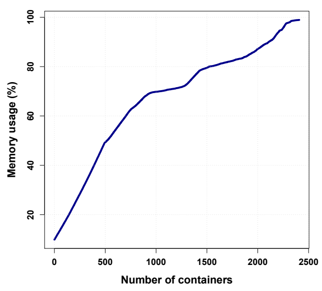
3.1 Scaling up the number of deployed containers within a PI

To scale up the containers, the docker image could be prepared as small as possible to minimize memory footprint. Consequently, the memory allocation for kernel to handle a web server process can be optimized. For this, we use a nano web server image[2] developed in assembly code in which size is less than 90 KB. To benchmark the capability of a PI (PI 2), we base our evaluation on memory consumption, CPU utilisation and the creation time (time taken to create a container) by using *sysstat*. In our first attempt, we were able to spin up only 37 containers, even though the memory usage and CPU utilisation were only 40% and 18% respectively. We then hacked the Docker daemon, to scale up the deployed containers to 2408. The procedures that we used are described in the full paper [7].

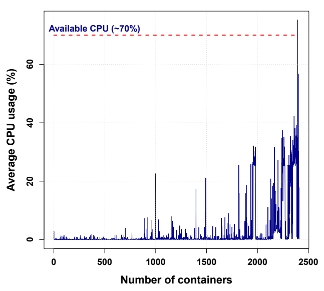
The results for our optimised docker are depicted in Figure 2 where we were able to scale up the number deployed containers to 2408. Specifically, memory usage is a key factor that limits the capability of running the containers on a PI. As shown in Figure 2a, the memory usage increases gradually when a container is added. The initial memory usage before creating the first container was about 98 KB (9.89%). We hit the limit of 2408 containers where there is no space for available memory.

Figure 2b shows the average utilisation of the quad-core CPU of the PI. Over the first thousand of deployed containers, the average CPU usage is very low (about 0.2%) with a few spikes. However as expected, the CPU usage increases significantly in the high load state (when the number of containers is larger than 2000). The quadcopter consumes about 30% of CPU resources. The available space can be around 70% which is sufficient to provide multiple services

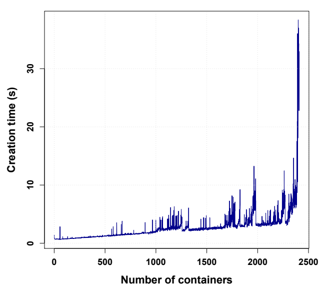
[2]https://github.com/hypriot/rpi-nano-httpd



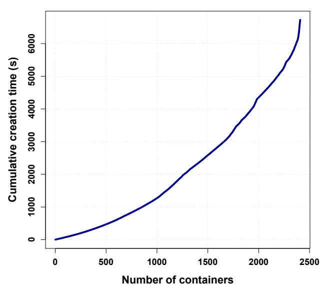
(a) Memory usage



(b) CPU usage

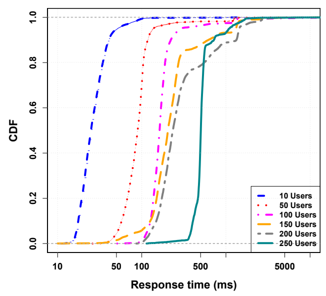


(c) Creation time

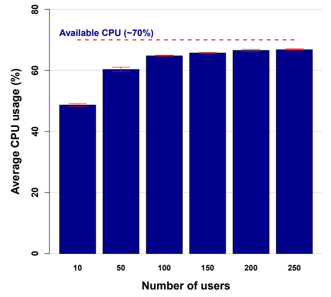


(d) Cumulative creation time

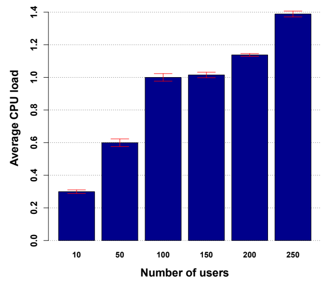
Figure 2: Number of web servers on a single PI



(a) CDF response time



(b) CPU utilization



(c) CPU load

Figure 3: Stress test with multiple requests

with Docker containers. Hence when the quadcopter is flying, we will not be able to have 2408 containers running in parallel. However, a *Cloudrone* can still support a large number of concurrent containers.

Figure 2c depicts the results of creation time where each point denotes the time it took for the n^{th} container to start up. The creation time is varied from 0.62 s to 38.37 s depending on the current CPU load and memory usage. In addition, we also plot the cumulative creation time of the 2408 containers (Figure 2d). The PI spent approximately 1 Hr and 50 minutes to spin out 2408 containers. On average each container requires 2.79 s to start up a web server.

Key takeaway message: A single PI can support significant amount of concurrent lightweight services.

3.2 Scaling up the number of users accessing a single service

In order to investigate the feasibility of using lightweight containers as a platform for the *Cloudrone*, we aim to evaluate the scalability of each of these containers running on a single PI while serving a large number of requests. We deploy a minimal static web server using httpd docker base image using a similar configuration as the experiments in the previous section. The benchmarking scenario represents the *Cloudrone's* operation where users on the ground can access the services provided by the *Cloudrone* through a wireless interface. Using the *Ab* - Apache HTTP server benchmarking tool[3], we conduct stress tests on the *Cloudrone* while scaling the number of concurrent users from 10 to 250. The total number of requests was set as 10000 transactions per experiment. For instance, in case of 10 users, 1000 requests were sent by each user. The measured RTT via ping tests with a clear line of sight at 50 feet was between 8ms-10ms.

Figure 3a illustrates the CDF of response time from the web container running on PI while varying the number of simultaneous users accessing the service. As shown in the figure, a single deployed docker container is capable of serving a large number of concurrent users. As expected, the average response time increases when the number of concurrent users is scaled up. Figure 3b and 3c plot the CPU utilisation and CPU load of the PI using *sysstat* tools. The CPU utilisation increases almost 20% when the number of users increase from 10 to 250. This increases the response time for the processes. Even though the utilization has not reached the capacity of CPU, processes still run slower as the CPU load increases. Figure 3c shows the average CPU load of the PI (sampled in one min intervals). As the arrival rate of user requests increases, the amount of computational work need to process also increases. This has impact on the response of time(Figure 3a).

Key takeaway message: A Docker container running on a single PI can support significant amount of concurrent users.

4. DISCUSSION

4.1 Scalability Challenges

Our preliminary benchmarking demonstrates that the PI is capable of functioning as a great micro cloud platform. Our benchmarking was carried out using a lightweight web server serving a lightweight webpage. It is important that the scalability of the *Cloudrone* should also be tested with heavier web servers serving applications such as Openstreetmaps. Each application will have different memory and CPU requirements and hence the number of containers

[3]https://httpd.apache.org/docs/current/programs/ab.html

that can be instantiated will vary depending on the type of applications which directly influences the size of the container (e.g., packages, data, library).

Another scalability challenge is to support a larger number of users within an area. Increasing the number of users causes an adverse influence on the response time which will cause service degradation. We envision, the different services will be provided by a swarm of drones and hence appropriate load balancing using techniques such as application layer anycast [10] could be used.

4.2 Service Retrieval

Cloudrones have two main challenges in terms of providing a reliable service. First, mobility poses a critical challenge. During mobility, ongoing sessions may break and sessions need to be reestablished. Second, considering the distributed nature of the services across a mesh of drones, identifying the location of the service across a mobile adhoc network is challenging. To solve the latter, techniques such as Multicast DNS (mDNS) [1] or application layer anycast could be used [10]. Another potential way to mitigate the problem of mobility and service discovery is to explore new architectures that utilise Information Centric Networking) [4]. ICN architectures such as Named Data Networking (NDN) [4] or SCANDEX [8] decouple the content from the location thus removing the need for the current end-to-end client server model such that the service and/or content can be served directly by any host that currently has the service/content. ICN thus integrates the provisioning of content with the locationless notion of information delivery in ICN allowing different flavours of caching, from on-path caching to edge caching through a farm of surrogate micro servers running on the *Cloudrones* that can be quickly integrated into the overall (ICN) routing fabric without the need for DNS redirection or other solutions of the current Internet. This inherently addresses the issues of mobility and reliability. As ongoing work, we are in the process of developing an ICN architecture for *Cloudrones* via the EU UMOBILE project [4].

4.3 Deployment Issues

Although *Cloudrones* demonstrate excellent potential for deploying localised service infrastructures in areas where access to services are crucial but are beyond reach - there are still major challenges that need to be surmounted.

Drones such as quadcopters have reduced flight times due to battery life. We envision this situation will change in the near future with better innovations in battery design and production or innovations in alternate sources of power e.g. hydrogen powered drones have flight times upto two hours [5]. *Cloudrones* also need not be in the air for their entire flight duration to provide access to its services. We envision that *Cloudrones* can be flown to an area and then can provide it's localised services from the ground (ideally powered by an energy source on the ground).

There are tight regulations in flying drones such as quadcopters. These rules have been laid out by Civil Aviation Authority (in the case of UK)[6]. Hence these rules should be adhered to and in some cases may be restrictive e.g. land or fly in a congested area. However, we believe, *Cloudrone* deployments will fall under the commercial aerial work, and hence special permission from the aviation authority will be required to fly.

[4]http://www.umobile-project.eu/

[5]http://www.bbc.co.uk/news/technology-35890486

[6]https://www.caa.co.uk/drones/

5. CONCLUSIONS

This paper presents *Cloudrone*- a preliminary idea of deploying an adhoc lightweight micro cloud infrastructure in the sky using indigenously built low cost drones, single board computers and lightweight Operating System virtualization technologies. We describe an initial design of the *Cloudrone* and provide a preliminary evaluation of the proposed system mainly focussed on the scalability issues of supporting multiple services and users. As part of future work, we plan to conduct large scale evaluation trials benchmarking the *Cloudrone* performance while in the air (in terms of throughput, latencies and energy) across a wide set of scenarios. We are also in the process of integrating Docker with NDN and performance benchmarks will be carried out. Finally, the current *Cloudrone* design does not fly autonomously and hence is strictly limited in terms of distance it can cover without manual intervention. As part of future work, we intend to build autonomous flying capabilities.

Acknowledgements

This work was supported by the EU H2020 UMobile project (Grant agreement no: 645124).

6. REFERENCES

[1] S. Cheshire and M. Krochmal. Multicast DNS. *Internet Engineering Task Force RFC 6762*, Feb. 2013.

[2] S. Corson and J. Macker. Mobile ad hoc networking (MANET): routing protocol performance issues and evaluation considerations. *Internet Engineering Task Force RFC 2501*, Jan. 1999.

[3] M Gowda, N Roy, and R Choudhury. Infrastructure mobility: A what-if analysis. HotNets-XIII, 2014.

[4] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard. Networking named content. CoNEXT '09, 2009.

[5] L. Li, T. Tang, and W. Chou. A rest service framework for fine-grained resource management in container-based cloud. In *IEEE CLOUD*, June 2015.

[6] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft. Unikernels: Library operating systems for the cloud. *SIGPLAN Not.*, 48(4):461–472, March 2013.

[7] A. Sathiaseelan, A. Lertsinsrubtavee, A. Jagan S, P. Baskaran, and J. Crowcroft. Cloudrone: Micro Clouds in the Sky. *ArXiv: 1604.08243*, April 2016.

[8] A. Sathiaseelan, L. Wang, A. Aucinas, G. Tyson, and J. Crowcroft. Scandex: Service centric networking for challenged decentralised networks. In *Mobisys DIYNetworking '15*, 2015.

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

[10] E. W. Zegura, M. H. Ammar, Zongming Fei, and S. Bhattacharjee. Application-layer anycasting: a server selection architecture and use in a replicated web service. *IEEE/ACM Transactions on Networking*, 8(4):455–466, Aug 2000.