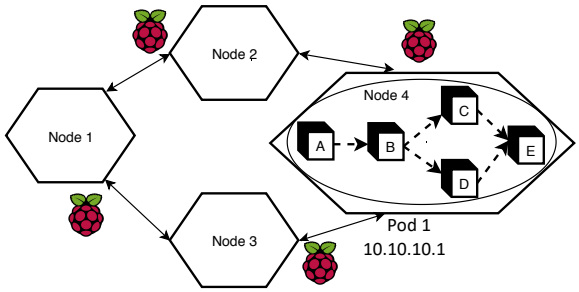
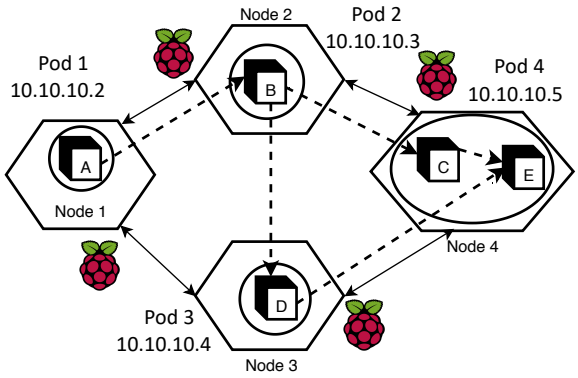
of fog nodes, it is often not possible to deploy the entire application on a single fog node. Therefore, the application needs to be distributed over the network of communicating fog nodes as shown in Figure 1(b). Deploying communicating containers on different fog nodes leads to data exchange between the fog nodes over the network which may incur some latency.



(a) Application on a single fog node.



(b) Application distributed over fog network.

Figure 1: Distribution of IoT Application.

We implement the design by utilizing the feasibility results from [5], where authors used raspberry pi devices to implement a fog network and deployed Docker containers for IoT applications. We use a video streaming IoT application that requires real-time, low latency services. Further, we use Kubernetes as an orchestration and monitoring tool to automate the deployment of application containers across a cluster of raspberry Pis. *pods* and *nodes* are two features of Kubernetes that are essential to the comprehension of our work. A pod is the basic control and management unit of Kubernetes consisting of one or more containers. Each pod is assigned a unique IP address. Kubernetes nodes are physical devices and run services like Docker, kubelet, and kube-proxy, necessary to run pods. Kubernetes architecture has a master node and worker nodes. The master node manages the Kubernetes cluster, and it is the entry point for all the administrative tasks. A worker node runs the application pods and is controlled by the master node. Next, we describe our experimental environment that compares the performance results in two scenarios shown in Figure 1.

3 IMPLEMENTATION DETAILS

Our testbed consists of a PC (Intel x86 architecture) serving as Kubernetes master node and four raspberry pi 3 (32-bit arm architecture) acting as worker nodes and forming a network as shown in Figure 1. All devices have Ubuntu 18.04 LTS as the operating system and run Kubernetes v1.17.3 and Docker-engine v19.03.6. The installation instructions for configuring raspberry Pis are available on GitHub at [3]. Kubernetes master node initializes a cluster and generates a token that is used by worker nodes to join the cluster. We use a video streaming nodeJS application (node v13.5.0) with five microservices communicating as shown in Figure 1. Each microservice of the application is containerized as a multi-platform image using Docker Buildx CLI plugin and pushed to the docker hub repository.

A Kubernetes pod is defined as a YAML file that consists of the details of one or more container images. Kubernetes runs a scheduling algorithm that assigns nodes to the pods. For the deployment strategy depicted in Figure 1(a), we place all the Docker containers of the application in the same pod and the pod is deployed on a single raspberry pi. Every container in a pod shares the network namespace, including the IP address and network ports and communicate with one another using the localhost without any network delay. On the other hand, in order to implement the placement depicted in Figure 1(b), the containers distributed on different Kubernetes nodes are placed in different pods. Kubernetes assigns a different IP address to each pod and communication between two pods placed on two different physical devices is performed over the network. We use Weave Net[4] to create a virtual network that connects Docker containers deployed across multiple nodes and enables their automatic discovery. This deployment strategy introduces a network delay in achieving the same functionality. We further use Kubernetes services that bind a port to exposes the pod. This allows us to access the pod on the Nodes IP address by sending a REST request.

4 DEMONSTRATION OVERVIEW

We start the streaming of video packets from container A and packets follow the path as shown in Figure 1. We will observe the output of the streaming on container E by sending an HTTP request to node 4. The demonstration will show that the functionality of the application is not affected by the distribution of communicating containers in separate pods placed on different raspberry pis. In the demo, we will show the quality of video streaming achieved with the two deployment strategies and the time lag experienced in the second scenario (Figure 1(b)). We also show how the application functionality, in terms of time delay, is affected when we consider mobile nodes and change the distance between the devices.

REFERENCES

[1] 2020. Docker Engine [Online]. https://docs.docker.com/engine/
[2] 2020. Kubernetes: Production-Grade Container Orchestration [Online]. https://kubernetes.io/
[3] 2020. Setup Instructions [Online]. https://github.com/paridhika/streaming-app
[4] 2020. Weave Net addon for Kubernetes [Online]. https://www.weave.works/docs/net/latest/kubernetes/kube-addon/
[5] Paolo Bellavista and Alessandro Zanni. 2017. Feasibility of Fog Computing Deployment Based on Docker Containerization over RaspberryPi. In *Proc. of ICDCN*.
[6] B. Butzin, F. Golatowski, and D. Timmermann. 2016. Microservices approach for the internet of things. In *Proc. IEEE ETFA*. 1–6.
[7] P. Kayal and J. Liebeherr. 2019. Autonomic Service Placement in Fog Computing. In *IEEE WoWMoM*. 1–9.
[8] P Kayal and J. Liebeherr. 2019. Distributed Service Placement in Fog Computing: An Iterative Combinatorial Auction Approach. In *IEEE ICDCS*. 2145–2156.
[9] OpenFog Consortium Architecture Working Group. 2017. OpenFog Architecture Overview White Paper. (2017).