Information-Centric Multi-Access Edge Computing Platform for Community Mesh Networks

Adisorn Lertsinsrubtavee
University of Cambridge
Cambridge, UK

Mennan Selimi
University of Cambridge
Cambridge, UK

Arjuna Sathiaseelan
Ammbr Research Labs
Cambridge, UK

Llorenç Cerdà-Alabern
UPC
Barcelona, Spain

Leandro Navarro
Ammbr Research Labs and UPC
Barcelona, Spain

Jon Crowcroft
University of Cambridge
Cambridge, UK

ABSTRACT

Edge computing is shaping the way to run services in the Internet by allowing the computations to become available close to the user's proximity. Many implementations have been recently proposed to facilitate the service delivery in data centers and distributed networks. However, we argue that those implementations cannot fully support the operations in Community Mesh Networks (CMNs) since the network connection is highly intermittent and unreliable. In this paper, we propose PiCasso, a novel multi-access edge computing platform that combines the advances in lightweight virtualisation and Information-Centric Networking (ICN). PiCasso utilises in-network caching and name based routing of ICN to optimise the forwarding path of service delivery. We analyse the data collected from Guifi.net, the biggest CMN worldwide, to develop smart heuristic for the service deployment. Through a real deployment in Guifi.net, we show that our service deployment heuristic HANET (HArdware and NETwork Resources), improves the response time up to 53% and 28.7% for stateless and stateful services respectively. Finally, using PiCasso for the service delivery in Guifi.net, we achieve 43% traffic reduction compared to the traditional host-centric communication.

1 INTRODUCTION

Community Mesh Networks (CMNs) are self-managed, large-scale networks that are built and organized in a non-centralized and open manner. As participation in these networks is open, they grow organically, since new links are created every time a host is added. Because of this, the network presents a high degree of heterogeneity with respect to devices and links used in the infrastructure and its management. Due to the large and irregular topology [47], highly skewed bandwidth/traffic distribution [31] and high software and hardware diversity [46] in the CMNs, the provisioning of the services is far from simple. Unfortunately, the current architectures and platforms in CMNs do not capture the dynamics of the network, and therefore they target best-effort service, disregarding QoS objectives [41]. This brings to CMNs the challenge to build infrastructures that support lightweight multi-tenancy at the network edge by allowing flexible hosting and fast delivery of local services.

The latest advances in lightweight virtualisation technologies (e.g., Docker [2], Unikernels [18]), allow many developers to build local edge computing platforms that could be used to deliver services within CMNs [19] [38]. Although delivering these lightweight services within a data center is trivial, delivering them across intermittent connectivity of CMNs comes with lot of challenges [39]. As a matter of fact, most of the edge computing platforms still rely on the host-centric communication that binds the connection to the fixed entity. The host-centric approach however struggles for *service delivery* i.e., transporting the service instances to the network edge as the connectivity can fail at any time [21, 36]. In addition to that, those platforms do not have specific strategy for the *service deployment* in CMN environment. This raises several questions about which services and where to host them, to deliver satisfactory performance to end users.

On the other hand, Information-Centric Networking (ICN) [49] has recently emerged as a potential solution for delivering named content. Instead of using IP address for communication, ICN identifies a content by name and forwards a user request through name-based routing. This decouples the content from its original location, where content can be delivered from any host that currently has it in its local storage [26]. Although ICN brings a lot of flexibility in terms of content delivery, the current ICN implementations are rather focused on simple static content (e.g., short message, video

COMPASS '18, June 20–22, 2018, Menlo Park and San Jose, CA, USA

ACM ISBN 978-1-4503-5816-3/18/06…$15.00
https://doi.org/10.1145/3209811.3209867

file). In this essence, we argue that ICN should be extended to better support the service delivery in edge computing.

To overcome the above-mentioned challenges, we present *PiCasso*, a unified edge computing platform that brings together the lightweight virtualisation technologies and a novel ICN paradigm to support both service delivery and service deployment in the CMN environment. We underpin PiCasso with a Docker container-based service that can be seamlessly delivered, cached and deployed at the network edge. The core of PiCasso is the *decision engine* component that deploys services on the basis of the service specifications and the status of the resources of the hosting devices. Unlike other edge computing platforms, PiCasso creates a new service abstraction layer using ICN to enable more flexibility in service delivery. Instead of hosting services in a fixed centralised location (e.g., service repository), PiCasso benefits from name-based routing and in-network caching capabilities of ICN by allowing edge devices to retrieve services from the nearest caches. Furthermore, PiCasso is also integrated with a service controller and full functional monitoring system to optimise service deployment decisions in CMNs. Specifically, our key contributions are summarized as follows:

First, we characterise the network performance of Guifi.net in the urban area of Sants in Barcelona (qMp Sants). Based on that, we identify the key performance indicators (e.g., node availability, bandwidth, CPU load) to be used in node selection and our service deployment heuristic.

Second, inspired by these characteristics, we propose PiCasso to facilitate the service deployment process while considering multiple QoS requirements and network constraints.

Third, we deploy PiCasso in Guifi.net and quantify the performance of the platform with multiple services. To the best of our knowledge, this is the first ICN deployment in a production wireless CMN. Through our extensive evaluations, we show how PiCasso can improve end-user experience (e.g., low latency, great responsiveness) using the HANET service deployment heuristic. On top of that, we prove that PiCasso is more efficient in terms of bandwidth consumption by using its ICN capabilities.

2 A COMMUNITY MESH NETWORK

The qMp (Quick Mesh Project) [21] has developed software tools to facilitate the deployment of community wireless mesh networks. It started in 2009 in the city of Barcelona, Spain. The QMPSU mesh is part of the Guifi.net community network, and the largest of the 10 qMp-based urban meshes in several districts of Barcelona. At the time of this writing, QMPSU (qMp network in the Sants district and UPC) has 86 operating nodes. In the network, there are two gateways that connect QMPSU to the rest of Guifi.net and the Internet.

In terms of hardware, mesh nodes have an outdoor router with a WiFi interface on the roof. These outdoor routers are used to build P2P (point-to-point) links in the network. The routers are connected through Ethernet to an indoor AP (access point) as a premises network where the edge services are running (e.g., on home-gateways, Raspberry Pi's etc.). Mesh routers use the BMX6 mesh routing protocol [32]. For our experimental cases, we attach (i.e, deploy) Raspberry Pi's to the routers in the network and use them as servers.

Methodology and data collection: We have collected network data by connecting via SSH to each QMPSU router and running basic system commands available in the qMp software distribution. Live measurements have been taken hourly during the entire month of September 2017. Our live monitoring system is operational and can be seen in [11]. Further, the data collected is publicly available. We use this data to analyse the main aspects of the QMPSU network.

2.1 QMPSU Network Characterisation

Node Characteristics: Figure 1 shows the Empirical Cumulative Distribution Function (ECDF) of the node availability collected for a period of one month. We define the availability of a node as the percentage of times that the node appears in a *capture*. A capture is an hourly network snapshot that we take from the QMPSU network (i.e., we took 744 captures in total). Figure 1 reveals that 10% of the nodes have availability lower than 90%. In a CMN such as QMPSU, users do not tend to deliberately reboot the device unless they have to perform an upgrade, which is not very common. Hence, the percentage of times that node appears in a capture is a relatively good measure of the node availability due to random failures (e.g., electric cuts, misconfiguration). Figure 2 shows the node out-degree in the network. Figure 2 reveals that on average, around 90% of the nodes in the network have more than 2 links and around 40% of the nodes have at least 5 links with an overall average degree of 6.9. This shows that the network is well-connected.
Network Performance: First, we characterize the wireless links of the QMPSU network by studying their bandwidth. Figure 3 shows the average bandwidth distribution of all the links. The figure shows that the link throughput can be fitted with a mean of 11.7 Mbps. At the same time Figure 3 reveals that the 60% of the nodes have 10 Mbps or less throughput. In order to measure the link asymmetry, Figure 4 depicts the bandwidth measured in each direction. A boxplot of the absolute value of the deviation over the mean is also depicted on the right. The figure shows that around 25% of the links have a deviation higher than 40%. After performing some measurements regarding the signaling power of the devices, we discovered that some of the community members have re-tuned the radios of their devices (e.g., transmission

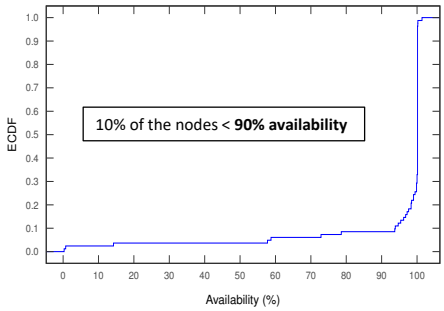


Figure 1: Node availability

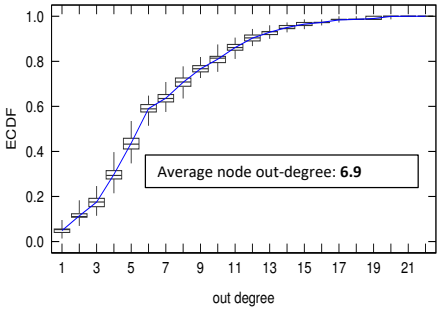


Figure 2: Node out-degree

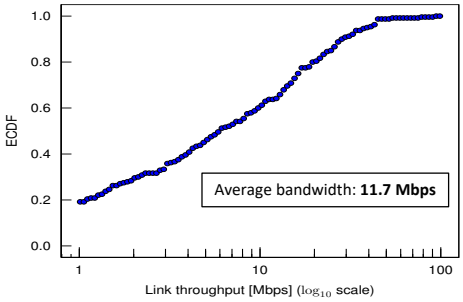


Figure 3: Bandwidth distribution

power, channel and other parameters), trying to achieve better performance, thus, changing the characteristics of the links.

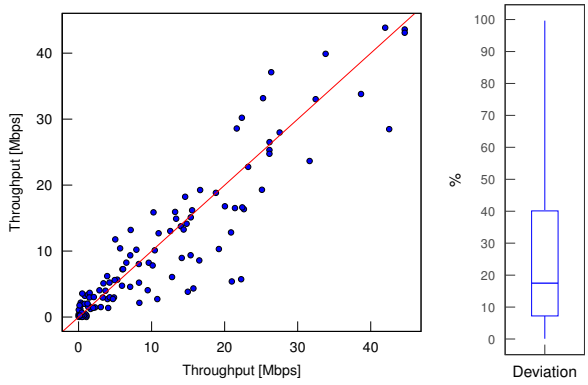


Figure 4: Bandwidth asymmetry

2.2 Key Observations

The measurements in the QMPSU network result in several observations:

Absence of the service-enabler platforms: Despite achieving the sharing of connectivity, Guifi.net and qMp-based CMNs have not been able to widely extend the sharing of ubiquitous cloud services, such as private data storage and backup, instant messaging, media sharing, social networking etc., which is a common practice in today's Internet through cloud computing. There have been efforts to develop and promote different services and applications from within community networks through community micro-clouds [20] but without widespread adoption. *Currently, there is no open source platform to bootstrap and manage decentralized community services. We claim that platforms that are easy to use, reliable, efficient and with smart decision making algorithms, can definitely boost the adoption of local services in the network.*

Dynamic topology: The QMPSU network is continuously changing and diverse due to many reasons, e.g., its community nature in an urban area; its decentralized organic growth with quite diverse technological choices for hardware, link protocols, channels; its mesh topology, the use of unlicensed, shared and noise-polluted spectrum; atmospheric fading. The network is not scale-free. The topology evolves organically and clearly differs with respect to centrally-planned conventional ISP networks. *There is need for fast, adaptive and effective heuristics that can cope with the topology dynamics.*

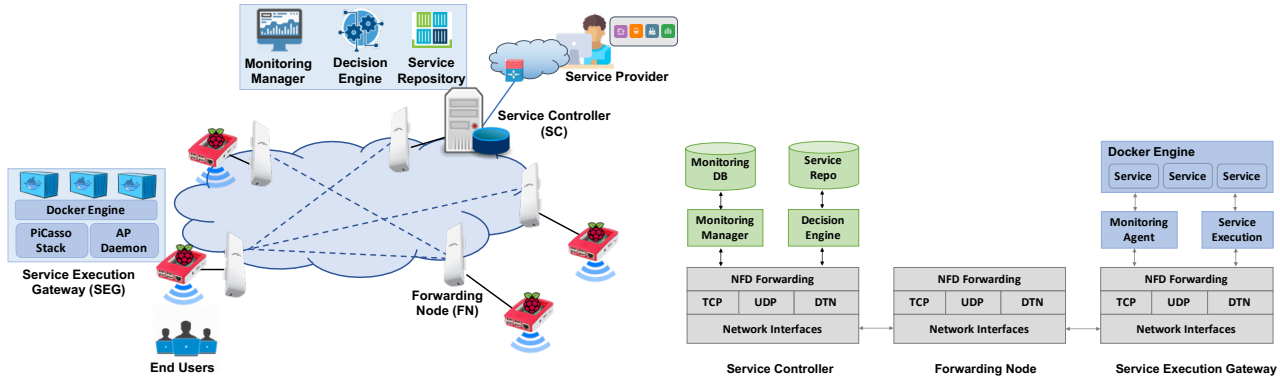
Non-uniform resource distribution: The resources are not uniformly distributed in the network. Wireless links have asymmetric quality. There is a highly skewed bandwidth, traffic and latency distribution. The current organic placement scheme in QMPSU and Guifi.net in general, is less than optimal, failing to follow the dynamics of the network and therefore cannot ensure levels of QoS. *The symmetry of the links, a frequent assumption in related work, is not that applicable for our case and algorithms (heuristics) unquestionably need to take this into account.*

3 PICASSO PLATFORM

To overcome the challenges in CMNs, PiCasso is implemented based on the service and access abstraction where lightweight virtualisation services are delivered through an ICN overlay. There are several ICN implementations [9, 10, 12, 13, 26] have been proposed during the past decade. Among those implementations, Named Data Networking (NDN) [26] is the most suitable candidate for PiCasso as it uses a simple stateful forwarding plane to utilise the distributed in-network caching without any controlling entity.

3.1 System Overview

The overview of PiCasso platform is presented in Figure 5a. The key entity is referred to *Service Controller* (SC) that periodically observes the network topology and resource consumption of potential nodes for the service deployment. In our model, we assume that the service providers upload their services to a service repository inside the SC before distributing to the network edge. To achieve the QoS and overcome the network connectivity problems, SC augments the monitoring data along with service deployment algorithms



(a) The overview of the PiCasso platform **(b) PiCasso's function blocks**

Figure 5: The architecture of PiCasso

to decide where and when to place the services. We also introduce the *Service Execution Gateway* (SEG) which provides a virtualisation capability to run a service instance at the network edge (e.g., users' house). In PiCasso, we use Docker, a container-based virtualisation to build lightweight services and deploy across the SEGs. Each SEG is also equipped with the access point daemon (e.g., hostapd [4]) to act as the point of attachment for the end-users to access the services via WiFi connection. A prototype of SEG has been developed on the Raspberry Pi 3 running the Hypriot OS Version 1.2.03 [5]. The *Forwarding Node (FN)* is responsible for forwarding the requests towards the original content source or nearby caches. Each FN is equipped with a storage while dynamically caching the content chunks that flow through it. Notice that, FN does not necessarily need to execute the services.

3.2 System Architecture

PiCasso's architecture is presented in Figure 5b which contains the function blocks of each entity (i.e., SC, SEG, and FN). Currently, PiCasso is written in Python, and implemented on top of NDN protocol stack and Docker [28].

NFD forwarding plane sits between the application and transport layers while looking at the content names and opportunistically forwarding the requests to an appropriate network interface. It creates an ICN overlay to support name-based routing over the network. We integrate the NFD forwarding plane to PiCasso architecture through a python wrapper of NDN APIs, called PyNDN [8]. The NFD maintains three types of data structure: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). FIB table maintains name prefixes with the outgoing interfaces based on routing protocols (e.g., static, NLSR [25]) and forwarding strategies(e.g., broadcast). PIT keeps track of the Interest requests that have already been forwarded by recording incoming faces and name of Interest messages. CS is a local cache integrated in every NFD node.

We have also extended the NDN protocol stack by introducing a DTN face. This new face communicates with an underlying DTN implementation that handles intermittence by encapsulating Interest and Data packets into a DTN bundle. The details of implementation and evaluation is available in [35, 36].

Service Execution runs on the SEG and has major functionalities as follows: registers the SEG to the service controller, receives push command to instantiate and terminate services dynamically regarding the decision of service deployment. This module uses docker-py [1], a python wrapper for Docker to expose the controlling messages to Docker engine.

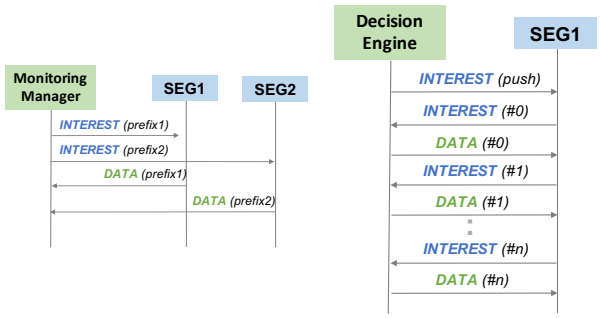
Monitoring Agent reports the SEG's status to SC by considering two layers of measurements. First, it measures the underlying hardware resources such as current memory usage, CPU utilisation and load. Second, it associates with Docker engine to report the status of running containers (e.g., container names) and resource consumption inside each container (e.g., CPU and memory usage).

Decision Engine (DE) is the core component of PiCasso that can make autonomic decisions of service deployment based on the combination of various measurement metrics such as resource consumption of underlying hardware, network topology, and service requirement. The DE also contains the algorithm repository where the service deployment algorithms can be dynamically updated regarding different deployment scenarios and service level agreements.

Service Repo is a repository storing dockerized compressed images. Our implementation allows the third party service providers to upload their service along with a deployment description augmented with specifications and QoS requirements. This description is written in JSON format.

Monitoring Manager periodically collects the monitoring data from each SEG and stores in the database (Monitoring

Figure 6: PiCasso Monitoring dashboard



(a) Pull-based model **(b) Push-based model**

Figure 7: Key operations in PiCasso. *(a)* Monitoring Manager retrieves the monitoring data from SEGs. *(b)* Decision Engine delivers the service to the SEG

DB). It is implemented based on a time series database, called InfluxDB [6]. We also implemented the dashboard for monitoring system using Grafana [3] to visualise time series data for SEG's measurements and application analytics (Figure 6).

3.3 PiCasso's Operations

3.3.1 Collecting Monitoring Data. This operation follows the native pull-based communication model of NDN. As shown in Figure 7a, the monitoring manager places the pull requests towards SEG1 and SEG2 while configuring name-prefixes as */picasso/monitoring/SEG1/* and */picasso/monitoring/SEG2/* respectively. When the SEG receives this pull Interest message, it attaches the current monitoring data with JSON format to the Data message and forwards to the same path that Interest message (reverse path forwarding) came from by using information in the PIT. To avoid receiving outdated data from the caches, we set the data freshness to a small value (e.g., 10ms).

3.3.2 Decision Making for Service Deployment. PiCasso relies on smart service deployment algorithms that aim to maximise the QoS as well as the network resources. This operation is controlled by the DE where it dynamically selects the appropriate algorithm from the repository regarding the scenario and requirements of the network. The output of the algorithm is a list of selected nodes ready for the service deployment and instantiation.

We propose HANET (HArdware and NETwork Resources) heuristic algorithm, which is designed specifically for service deployment in the unreliable network environment such as CMNs. HANET uses the state of the underlying CMN (i.e., QMPSU) to optimize service deployment decision. In particular, it considers three sources of information: i) network bandwidth, ii) node availability, iii) hardware resources to make optimal decisions [40]. First, we test HANET with the static data obtained from the QMPSU network (i.e., bandwidth, availability, and CPU data). Then we ran HANET in a real CMN (i.e., QMPSU) and quantify the performance achieved after deploying real services. The HANET heuristic algorithm (see Algorithm 1) runs in three phases:

Phase 1 - Network Setup Phase: We initially build the topology graph of the QMPSU network. The network topology graph is constructed by considering only operational nodes, marked in "working" status, and having one or more links pointing to another node (i.e., we remove the disconnected nodes). Once the topology graph is constructed, we check the availability of the nodes in the network. The nodes that are under the predefined availability threshold (λ) are removed. Then, we use the K-Means partitioning algorithm to group nodes based on their geo-location. The idea is to get back clusters of nodes that are close to each other. The K-Means algorithm forms clusters of nodes based on the Euclidean distances between them, where the distance metrics in our case are the geographical coordinates of the nodes. Each cluster contains a full replica of a service, i.e., the algorithm in this phase partitions the network topology into k (maximum allowed number of service replicas) clusters. This is plotted as KMeans_C in the Figure 8.

Phase 2 - Computation Phase: This phase is based on the concept of finding the cluster *heads* maximizing ($\text{argmax}_C \sum_{i=1}^{k} \sum_{j \in Ci} B_{ij}$) the bandwidth ($B_{ij}$) between them and their member nodes in the clusters C_k formed in the first phase. The bandwidth between two nodes is estimated as the bandwidth of the link having the minimum bandwidth in the shortest path. The computed cluster heads are the candidate nodes for the service deployment.

Phase 3 - Content Placement Phase: After the cluster heads are computed in Phase 2, the services are placed on the selected cluster heads if their CPU load is under the predefined threshold (α). If this condition is satisfied, the service image is pulled from the Service Repo and pushed to the selected edge nodes (i.e., deployed and started). The CPU threshold can be set at the monitoring dashboard and the notification will be sent to the DE when the measured CPU load violates this threshold.

Algorithm 1 HANET Algorithm

Require: input = *qMpTopology.xml*
Rn ▷ availability of node n
λ ▷ availability threshold
CPUch ▷ CPU load of cluster head
α ▷ CPU threshold

Phase 1 – Network Setup Phase

1: **procedure** NETWORKSETUP(input)
2: g = *BuildTopology*(input)
3: g′ = *SanitizeGraph*(g)
4: **for** each *line* in *g′* **do** // *sanitization process*
5: Remove disconnected nodes
6: Ensure bidirectional links
7: Remove nodes with no metrics
8: **end for**
9: **return** *g′*
10: **if** Rn $\geq \lambda$ **then**
11: *PerformKMeans*(g′, k)
12: **return** *C*
13: **end if**
14: **end procedure**

Phase 2 – Computation Phase (Bandwidth Max.)

15: **procedure** COMPUTEHEADS(C)
16: clusterHeads ← *list*()
17: **for all** i ∈ Ck **do**
18: **for all** i ∈ Ck **do**
19: Bi ← 0
20: **for all** j ∈ *setdiff*(C,i) **do**
21: Bi ← Bi + *estimate.route.bandw*(g′,i,j)
22: **end for**
23: clusterHeads ← argmax$_C \sum_{i=1}^{k} \sum_{j \in Ci} B_{ij}$
24: **end for**
25: **end for**
26: **return** clusterHeads
27: **end procedure**

Phase 3 – Content Placement Phase (Hardware)

28: **procedure** PLACEMENTPHASE
29: **for** each clusterHeads **do**
30: **if** CPUch $\leq \alpha$ **then**
31: *DeployService*()
32: *StartService*()
33: **end if**
34: *GoForNextClusterHead*()
35: **end for**
36: **end procedure**

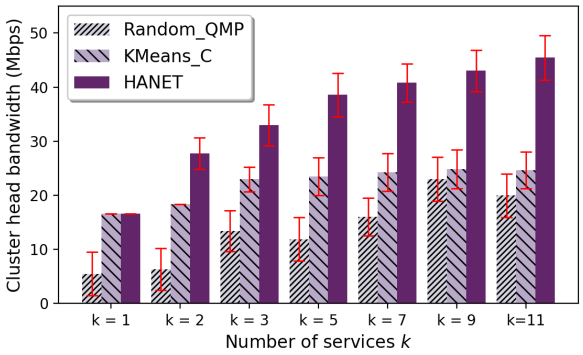


Figure 8: Average bandwidth to the cluster heads

3.3.3 Deliver Service to the Edge. When the DE retrieves a list of selected node names from the service deployment algorithm, it will start the service delivery process which requires the push-based communication model. However, the current implementation of NDN, it supports only the pull-based model where a consumer (i.e., SEG) has to initiate the communication. To support this operation, we have implemented the push communication model based on Interest/Data exchange of primitive NDN. We follow the publish-subscribe model [23] where a data producer (DE) publishes contents or services via Interest message to a subscribed consumer which in turn trigger an Interest back from the consumer to fetch the data. Figure 7b illustrates the Interest/Data exchange of the push-based model, where the DE initially sends a push Interest message to SEG1 with a name prefix: */picasso/service_deployment/push/SEG1/service_name*.

To distinguish the push Interest message from the NDN pull model, a name component, "push" is added after the operation name (i.e., "service_deployment"). Consequently, when SEG1 receives the push Interest message, it discards the ("push") and ("SEG_ID") prefixes while reconstructing a new Interest name: */picasso/service_deployment/service _name/#00* to request the service image. In NDN, a content is divided into several chunks, the last prefix is reserved for the requesting chunk ID which is started from zero (e.g., #00).

4 PICASSO DEPLOYMENT IN GUIFI.NET

In order to understand the feasibility of running the PiCasso platform and the possible gains of our service deployment heuristic HANET in a real production CMN, we deploy PiCasso in a real hardware connected to the nodes of a qMp-based network located in the Sants district of Barcelona (QMPSU). We have strategically deployed 10 SEGs to cover the area of QMPSU as presented in Figure 9. In our configuration, SEGs are connected to the mesh routers via Ethernet cable and the service controller is centrally set up inside the main campus of the UPC university where the Guifi.net research lab is located.

Node Selection: The location of the the five SEGs deployed is chosen based on the output of the HANET algorithm (i.e., highlighted with red circle in Figure 9). This corresponds to the top-ranked nodes (i.e., cluster heads) selected from the HANET; with higher bandwidth, availability and CPU resources. Based on this, we deploy five Raspberry Pi's to the selected mesh routers given by the HANET algorithm. The other five mesh routers in QMPSU are selected randomly for comparison purposes. In this set, we cover nodes with different properties: high degree centrality, nodes that are not well connected, nodes acting as bridges etc. All nodes are well-distributed in the QMPSU network.

ICN Overlay: We follow the ICN-as-an-Overlay approach [34] to construct the ICN shim layer on top of the existing QMPSU routing protocol (i.e., BMX6). The NFD forwarding plane is responsible for managing the name based routing in this ICN layer. In this deployment trial, we use a static routing to setup the forwarding table (FIB) of each SEG and service controller based on actual information taken from the IP routing table of mesh routers in the QMPSU network.

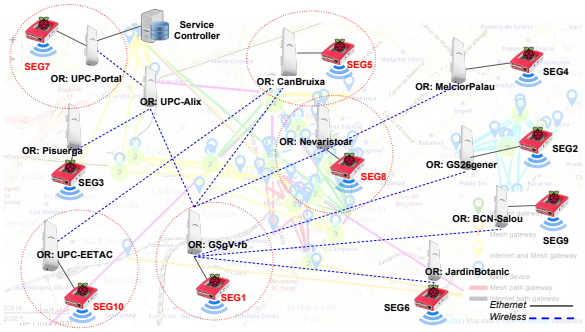


Figure 9: The topology of PiCasso deployment in QMPSU

5 PERFORMANCE EVALUATION

This section analyses the performance of PiCasso platform deployed in the QMPSU network. We concentrate on the benchmarking of two services: user and network-focused services. From the user services, we quantify the performance of the HANET heuristic using a stateless service (ApacheBench [14]) and a stateful Web2.0 service (Cloudsuite Web Serving benchmark [33]). The evaluation of end-user services is based on the web technology while the response time is the key performance metric. On the other hand, the evaluation of network services focuses on the efficiency of service delivery in PiCasso comparing with a traditional host-centric communication (HCN) approach.

5.1 Evaluation of End-user Services

Undoubtedly, deploying multiple service instances can significantly improve the QoS, since servers or containers can balance the load and response to user requests faster. However, in practice, it is not trivial to deliver a service instance in every location as it comes with extra costs such as bandwidth consumption, memory usage and CPU load. To balance this trade-off, we apply the HANET service deployment heuristic to decide where to place the services. We compare the HANET heuristic with the Random heuristic i.e., the existing in-place and naturally fast strategy in the QMPSU network.

5.1.1 Impact on Stateless User Services. In this evaluation, we focus on the response time of the HTTP requests while considering a different number of replicas (e.g., $k = 1$ and $k = 2$). The location of the replica is determined by the HANET algorithm using the measurements from the QMPSU dataset as well as the real-time monitoring data from the PiCasso platform. Based on HANET, $\{SEG1\}$ and $\{SEG1, SEG8\}$ are selected for $k = 1$ and $k = 2$ respectively, as highlighted in Figure 9. In this experiment, we consider a lightweight web server, namely *hypriot/rpi-busybox-httpd* which contains a static single HTML document with a link to a local jpeg image (the payload size is 304 bytes). This service image is delivered to the selected SEGs by using the operation in Figure 7b. To generate the HTTP requests, the Apache tool is run in all deployed 10 SEGs as client nodes. In each node, we configured the Apache tool to create a number of concurrent active users as 10, subsequently sending 500 HTTP requests in total to the closest replica.

Figure 10 illustrates the CDF of the response time collected from the Apache client nodes. Generally, HANET achieves *significantly* lower response times compared to the Random heuristic. We observed that, for $k = 1$, 80% of the requests achieve response time less than 360 ms when using HANET and 700 ms when using the Random, respectively. Furthermore, increasing the number of replicas to $k = 2$ also reduces the response time of both algorithms. By considering 80% of the requests, HANET reduces the response time up to 190 ms and Random up to 324 ms, that is about 47.22% and 53.71% improvement compared to $k = 1$ case. For HANET, $k = 2$ is quite sufficient as almost 90% of the requests can achieve the response time less than 500 ms which is widely acceptable for the static web application.

5.1.2 Impact on Stateful User Services. The second experiment is the Web 2.0 service which mimics a social networking application (e.g., Facebook). The content of the Web 2.0 website is dynamically generated from the actions of multiple users. For the evaluation, we use the dockerized version of the CloudSuite Web Serving benchmark [33]. Cloudsuite benchmark has four tiers: the web server, the database server, the memcached server, and the clients. Each tier has its own Docker image. The web server runs Elgg [7] social networking engine and it connects to the memcached server and the database server. The clients (implemented using

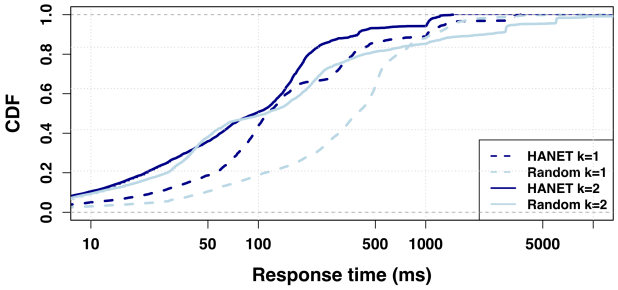


Figure 10: Response time of HTTP requests

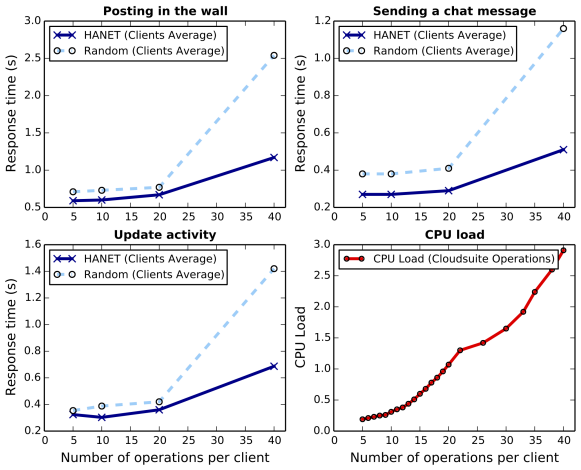


Figure 11: Cloudsuite Operations (HANET vs. Random)

the Faban workload generator) send requests to login to the social network and perform different operations.

We use 10 SEGs attached to the QMPSU mesh routers, where nine of them act as clients. One of the nodes is used to deploy the web server. The web server, database server, and memcached server are always collocated in the same host. On the clients side, we measure the response time while performing operations such as: posting on the wall, sending a chat message, updating live feed operation, etc. In Cloudsuite, each operation is assigned an individual QoS latency limit. If less than 95% of the operations meet the QoS latency limit, the benchmark is considered to be failed. The location of the web server, database server, and memcached server has a direct impact on the client response time.

Figure 11 depicts three Cloudsuite operations performed when placing the web server with the HANET and Random heuristic. Figure 11 reveals that HANET outperforms Random for all the operations; for PostingInTheWall operation the improvement brought by HANET is 26.4%, for SendChatMessage operation 35.7% and for UpdateActivity operation 24%. We can notice that the gain brought by HANET is higher for more intensive workloads (i.e., on average 53% improvement when performing 40 operations per client). Further, Figure 11 shows the average CPU load observed in the clients when performing a different number of operations. The figure reveals that for 40 operations per client, CPU is reaching a load of 3, and as a result of this we have higher response times.

5.2 Evaluation of Network Services

To evaluate PiCasso in terms of network services, we focus on service delivery capability while considering how service instances are made available at the network edge. We focus on the delivery cost which is the total time counting from when the DE makes a service deployment decision until the service is delivered to the SEG. We compare the delivery cost of our solution (PiCasso) with the classic host-centric networking approach (HCN) which is commonly used in many edge computing platforms such as Cloudy [20] and Paradrop [30]. To implement this approach, we disable in-network caching facility of PiCasso and direct the service to be delivered from the service repo to each SEG, which is also similar to the IP unicast.

5.2.1 Analysis of Service Delivery Cost. In this evaluation, we select four dockerised containers which have different image sizes from the docker hub (see details in Table 1) and migrate them from the service repo to all the deployed SEGs.

Image name	Size	HCN	PiCasso
hypriot/rpi-nano-httpd	88 kB	0.401 s	0.139 s
hypriot/rpi-busybox-httpd	2.16MB	2.566 s	1.014 s
armhf-alpine-nginx	14.95 MB	16.021 s	6.741 s
armbuild/debian	145 MB	154.94 s	70.741s

Table 1: Comparison of the average delivery cost

Overall, the average delivery cost achieved by PiCasso is substantially lower than the HCN approach. For instance, PiCasso can reduce the delivery cost of the *armbuild/debian* image from 154.94 to 70.74 seconds which is about 54% improvement compared to the HCN solution. To have a closer look how a service image is delivered, we focus on the Debian image and plot the delivery time across each node, as presented in Figure 12. By comparing HCN and PiCasso, we observe that every SEG is better off through the in-network caching and named-based routing capabilities of PiCasso. The SEGs running PiCasso are able to retrieve the data chunks from the nearest cache (discussion will be provided with Figure 13). On the other hand, the HCN approach is inefficient in terms of bandwidth utilisation. Given an example of SEG6, HCN acquires 295 second to deliver the service which is converted to 0.49 MBps throughput. However, from the iperf measurement, the bandwidth between SEG6 and the service repo is approximately 1.32 MBps. As previously stated in Section 2.1, the resources in the QMPSU network are not

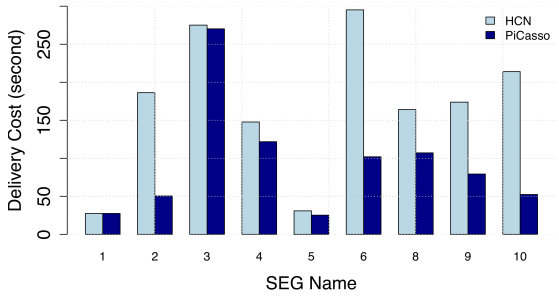


Figure 12: Inspecting the delivery cost of each SEG

uniformly distributed. This indicates that the traditional HCN approach is not sufficient to support good quality service delivery in this dynamic environment.

5.2.2 Investigating Traffic Consumption of Service Delivery. Previous results demonstrated that PiCasso efficiently improves the service delivery in the QMPSU network. To further investigate this, we perform sensitivity analysis on the amount of traffic that is consumed for delivering the service images to the SEGs. We inspect the amount of traffic among SEGs and the service controller from the *nfd-status* reports [15]. However, the information from these reports contains only the traffic of an overlay network. To construct the actual traffic that spread over the QMPSU network, we map the paths from PiCasso overlay with the routing tables of BMX6 routing protocol. For instance, the path between service controller and SEG5 (see Figure 9) can be mapped to *UPC-Portal - UPC-Alix - GSgV_rb - GSgranVia - CanBruixa* (i.e., names denote the location of routers).

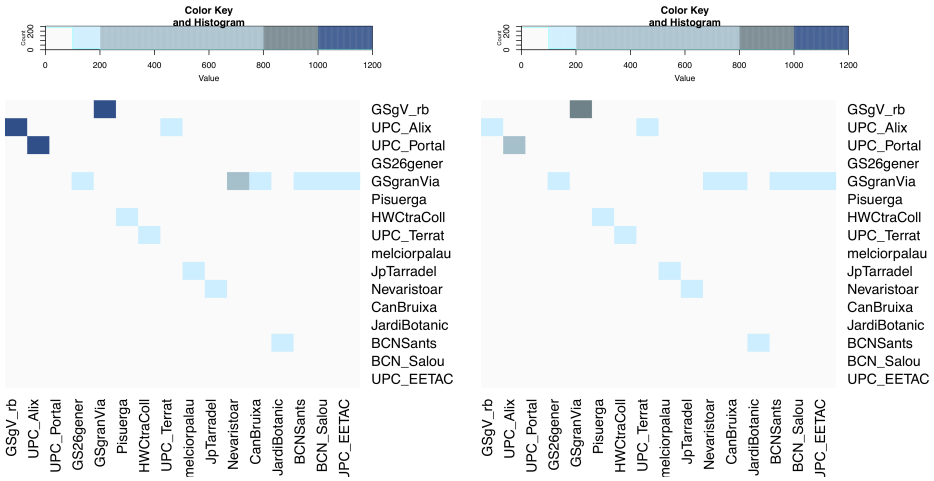
Figure 13 presents the distribution of data traffic sent among the mesh routers to deliver a service image to all 10 SEGs. Here, we solely present the results of delivering *armbuild/debian* image (the largest image size in the experiments) due to space constraints. The total amount of traffic consumed by HCN approach is approximately 5.375 GB while our PiCasso achieved only 3.05 GB which is about 43.24% reduction. In case of HCN, the most dominant traffic path is a link between *GSgV_rb* and *UPC-Portal* since this is a bottleneck link between nodes deployed in QMPSU and the service controller at the UPC Campus. In contrast, PiCasso significantly reduces the traffic over this link. The reason is that PiCasso takes benefits of the edge caching by allowing SEGs retrieve a service image from the closer node. As illustrated in Figure 9, we deployed SEG1 at the node *GSgV_rb* which has the highest degree centrality (i.e., it is well connected by other nodes). In this manner, several nodes (e.g., SEG2, SEG5, SEG6, SEG8, SEG9) can directly retrieve the data chunks from the cache of SEG1. This is very useful as the cache is utlised closer to the network edge.

6 RELATED WORK

PiCasso brings together many building blocks aiming at developing an efficient platform for service delivery in challenging network environments. From this aspect, we can classify three main related areas of work as follows:

Information Centric Networks: The clean slate approach called Information Centric Networks (ICNs) has recently emerged which inherently integrated the content delivery capability in the architecture [49]. Several research projects have been proposed to cope with the efficiency of content delivery, which have also been considered as the future Internet architecture [9, 10, 12, 13, 26]. Among those ICNs realisations, NDN [26] aims to utilise the widely distributed caching in the network by delivering contents based on name-based routing with a simple stateful forwarding plane. In contrast, PURSUIT [10] and RIFE [12] architectures are designed based on a centralised solution where there is a central entity to control the published and subscribed requests.

Edge Computing: Many researchers have leveraged the advantage of lightweight virtualisation technologies (e.g., Docker [2], Unikernel [18]) by proposing the edge computing platforms to improve the QoS, security and privacy [20, 24, 27, 30, 37, 39]. In [37], *Sathiaseelan et al.* propose Cloudrone, an edge computing platform for delivering services over a cluster of flying drones. However, this work reports only a feasibility study of the system and evaluation of scaling massive docker containers over a single Raspberry Pi. Similar to [24], *Yehia et al.* only study the scalability of docker containers with different generations of the Raspberry Pi. Accordingly, these works are still lacking vital components for edge computing platforms such as orchestration, monitoring and communication modules. The prototype of PiCasso has been introduced in [29]. However, the evaluation of communication protocol for delivering the service has not been discussed. Paradrop [30] is a specific edge computing platform that allows third-party developers to flexibly create new types of services. Cloudy [20] is the core software of the community clouds [41], as it unifies the different tools and services for the distributed cloud system with a Debian-based Linux distribution. The common limitation of these two platforms is lacking a service controller who automatically applies complex algorithms for service deployment regarding network condition and hardware resources. Furthermore, they rely on host-centric communication which is not efficient for CMNs as discussed in our results. Similar to our work is SCANDEX [39], a framework that brings together together the lightweight virtulisation, ICN and DTN technologies. However, the authors propose only the conceptual design architecture. NFaaS [27], is based on unikernel and NDN while enabling the seamless execution of stateless microservices across the network. However, the



(a) Experiments with HCN **(b) Experiments with PiCasso**

Figure 13: Data traffic distributed over the QMPSU network. X and Y axis denote the name of mesh routers while the gradient on each coordinate represents the density of traffic (MBytes) over a link between two routers.

authors only evaluate the system through simulation while the real implementation is still under development.

Service Placement: Al Arnaut in [16, 17], proposes a content replication scheme for wireless mesh networks. The proposed scheme is divided into two phases including the selection of replica nodes (network setup phase) and content placement, where content is cached in the replicas based on popularity. The work of Elmroth [45] takes into account rapid user mobility and resource cost when placing applications in Mobile Cloud Networks (MCN). Spinnewyn [42] provides a resilient placement of mission-critical applications on geo-distributed clouds using a heuristic based on subgraph isomorphism detection. Tantawi [43, 44] uses biased statistical sampling methods and hierarchical placement policies for cloud workload placement. Wang in [48] studies the dynamic service migration problem in mobile edge-clouds that host cloud-based services at the network edge. Coimbra in [22] proposes a novel service placement approach based on community finding (using a scalable graph label propagation technique) and decentralized election procedure. Most of the work in the data centers and distributed clouds consider micro-datacenters, where in our case the CMNs such as the QMPSU network consist of constraint/low-power devices such us Raspberry Pi's.

7 CONCLUSION

A particularity of CMNs is that they are heterogeneous in nature with a high level of node and network diversity, including different topologies. As a result, they face several technical challenges related to resource management, instability, and unavailability. In this paper, we have analysed the characteristics of the QMPSU Guifi.net CMN, to identify the key requirements for developing an edge computing platform. From the analysis, we argue that most of the existing platforms do not allow to achieve performance and quality objectives related to QoS since they rely on the Host-Centric Communication. In this aspect, we proposed PiCasso, a flexible edge computing platform that utilises the strength of the lightweight virtualisation technology and Information-Centric Networking (ICN) to overcome the challenges in CMNs. Unlike other platforms, PiCasso contains a Decision Engine that manages the service deployment operation in CMNs. We augmented the Decision Engine with a service deployment heuristic called HANET, which considers both hardware and network resources when placing services. Based on the results, HANET optimally selects the nodes to host the services and ensures that the end-users achieve an improved QoS. Apart from improving the QoS for end-users, our results show that ICN plays a key role in improving the service delivery time as well as reducing the traffic consumption in CMNs.

In future work, we intend to develop several algorithms (e.g., for different topologies) that could support diverse scenarios and requirements for service deployment. Furthermore, we wish to deploy PiCasso in other CMNs which might have different characteristics.

ACKNOWLEDGEMENTS

This paper has been supported by the AmmbrTech Group, the Spanish government TIN2016-77836-C2-2-R and EU Horizon 2020 under the projects RIFE (644663) and UMOBILE (645124). The authors would like to thank the people from the Guifi.net (QMPSU) community network for hosting the servers and supporting the experiments. Thanks to Musab Isah for proofreading the article.

REFERENCES

[1] [n. d.]. A Python library for the Docker Engine API. https://github.com/docker/docker-py. ([n. d.]). Accessed: 2018-02-10.

[2] [n. d.]. Docker technology. https://www.docker.com/what-docker. ([n. d.]). Accessed: 2018-02-10.

[3] [n. d.]. Grafana: The open platform for analytics and monitoring. https://grafana.com/. ([n. d.]). Accessed: 2018-02-10.

[4] [n. d.]. Hostapd: Host access point daemon. https://wiki.gentoo.org/wiki/Hostapd. ([n. d.]). Accessed: 2018-02-10.

[5] [n. d.]. Hypriot Docker Image for Raspberry Pi. https://blog.hypriot.com/downloads/. ([n. d.]). Accessed: 2018-02-10.

[6] [n. d.]. InfluxDB: The Time Series Database. https://www.influxdata.com/time-series-platform/influxdb/. ([n. d.]). Accessed: 2018-02-10.

[7] [n. d.]. Introducing a powerful open source social networking engine. https://elgg.org/. ([n. d.]). Accessed: 2018-02-10.

[8] [n. d.]. NDN client library with TLV wire format support in native Python. https://github.com/named-data/PyNDN2. ([n. d.]). Accessed: 2018-02-10.

[9] [n. d.]. NetInf - Network of Information. http://www.netinf.org. ([n. d.]). Accessed: 2018-02-10.

[10] [n. d.]. PURSUIT a Pub/Sub Internet. http://www.fp7-pursuit.eu/PursuitWeb/. ([n. d.]). Accessed: 2018-02-10.

[11] [n. d.]. qMp live monitoring. http://dsg.ac.upc.edu/qmpsu/index.php. ([n. d.]). Accessed: 2018-02-10.

[12] [n. d.]. RIFE: Architecture for an Internet for everybody. https://rife-project.eu/. ([n. d.]). Accessed: 2018-02-10.

[13] [n. d.]. Scalable and Adaptive Internet Solutions (SAIL). http://www.sail-project.eu. ([n. d.]). Accessed: 2018-02-10.

[14] [n. d.]. âĂIJApache Benchmarking toolâĂİ. ([n. d.]). https://httpd.apache.org/docs/2.4/programs/ab.html

[15] Alexander Afanasyev. 2018. *NFD Developer's Guide*. Technical Report. http://named-data.net/publications/techreports/

[16] Zakwan Al-Arnaout, Qiang Fu, and Marcus Frean. 2012. A Content Replication Scheme for Wireless Mesh Networks. In *Proceedings of the 22Nd International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV '12)*. ACM, New York, NY, USA, 39–44. https://doi.org/10.1145/2229087.2229098

[17] Z. Al-Arnaout, Q. Fu, and M. Frean. 2014. An efficient replica placement heuristic for community WMNs. In *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*. 2076–2081. https://doi.org/10.1109/PIMRC.2014.7136514

[18] Madhavapeddy Anil and David J. Scott. 2013. Unikernels: Rise of the Virtual Library Operating System. *Queue* 11, 11, Article 30 (Dec. 2013), 30:30–30:44 pages.

[19] Roger Baig, Roger Pueyo Centelles, Felix Freitag, and Leandro Navarro. 2017. On edge microclouds to provide local container-based services. In *2017 Global Information Infrastructure and Networking Symposium, GIIS 2017, Saint Pierre, France, October 25-27, 2017*. 31–36. https://doi.org/10.1109/GIIS.2017.8169801

[20] Roger Baig, Felix Freitag, and Leandro Navarro. 2018. Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons. *Future Generation Computer Systems* (2018). https://doi.org/10.1016/j.future.2017.12.017

[21] Llorenç Cerdà-Alabern, Axel Neumann, and Pau Escrich. 2013. Experimental Evaluation of a Wireless Community Mesh Network. In *Proceedings of the 16th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '13)*. ACM, New York, NY, USA, 23–30. https://doi.org/10.1145/2507924.2507960

[22] Miguel E. Coimbra, Mennan Selimi, A. P. Francisco, Felix Freitag, and Luís Veiga. 2018. Gelly-Scheduling: Distributed Graph Processing for Service Placement in Community Networks. In *33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018)*. ACM.

[23] Upeka De Silva, Adisorn Lertsinsrubtavee, Arjuna Sathiaseelan, Carlos Molina-Jimenez, and Kanchana Kanchanasut. 2016. Implementation and Evaluation of an Information Centric-based Smart Lighting Controller. In *Proceedings of the 12th Asian Internet Engineering Conference (AINTEC '16)*.

[24] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. RiviÃĺre. 2017. On Using Micro-Clouds to Deliver the Fog. *IEEE Internet Computing* 21, 2 (Mar 2017), 8–15. https://doi.org/10.1109/MIC.2017.35

[25] A K M Mahmudul Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. 2013. NLSR: Named-data Link State Routing Protocol. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking (ICN '13)*. ACM, New York, NY, USA, 15–20. https://doi.org/10.1145/2491224.2491231

[26] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. 2009. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09)*. ACM, New York, NY, USA, 1–12. https://doi.org/10.1145/1658939.1658941

[27] MichałKról and Ioannis Psaras. 2017. NFaaS: Named Function As a Service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking (ICN '17)*. ACM, New York, NY, USA, 134–144. https://doi.org/10.1145/3125719.3125727

[28] Adisorn Lertsinsrubtavee. Visited in April 2018. PiCasso's code repository. https://github.com/AdL1398/PiCasso. (Visited in April 2018).

[29] Adisorn Lertsinsrubtavee, Anwaar Ali, Carlos Molina-Jimenez, Arjuna Sathiaseelan, and Jon Crowcroft. 2017. PiCasso: A lightweight edge computing platform. In *Proceedings of the 6th IEEE International Conference on Cloud Networking (CloudNet'17)*.

[30] P. Liu, D. Willis, and S. Banerjee. 2016. ParaDrop: Enabling Lightweight Multi-tenancy at the NetworkâĂŹs Extreme Edge. In *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Vol. 00. 1–13. https://doi.org/10.1109/SEC.2016.39

[31] Leonardo Maccari and Renato Lo Cigno. 2015. A week in the life of three large Wireless Community Networks. *Ad Hoc Networks* 24 (2015), 175 – 190. https://doi.org/10.1016/j.adhoc.2014.07.016 Modeling and Performance Evaluation of Wireless Ad-Hoc Networks.

[32] A. Neumann, E. Lopez, and L. Navarro. 2012. An evaluation of BMX6 for community wireless networks. In *8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 I*. 651–658. https://doi.org/10.1109/WiMOB.2012.6379145

[33] Tapti Palit, Yongming Shen, and Michael Ferdman. 2016. Demystifying Cloud Benchmarking. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 122–132.

[34] A. Rahman, D. Trossen, D. Kutscher, and R. Ravindran. 2018. *Deployment Considerations for Information-Centric Networking (ICN)* . Internet-Draft. https://tools.ietf.org/id/draft-rahman-icnrg-deployment-guidelines-05.html#

[35] C. A. Sarros, S. Diamantopoulos, S. Rene, I. Psaras, A. Lertsinsrubtavee, C. Molina-Jimenez, P. Mendes, R. Sofia, A. Sathiaseelan, G. Pavlou, J. Crowcroft, and V. Tsaoussidis. 2018. Connecting the Edges: A Universal, Mobile-Centric, and Opportunistic Communications Architecture. *IEEE Communications Magazine* 56, 2 (Feb 2018), 136–143. https://doi.org/10.1109/MCOM.2018.1700325

[36] Christos-Alexandros Sarros, Adisorn Lertsinsrubtavee, Carlos Molina-Jimenez, Konstantinos Prasopoulos, Sotiris Diamantopoulos, Dimitris Vardalis, and Arjuna Sathiaseelan. 2017. ICN-based Edge Service

Deployment in Challenged Networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking (ICN '17)*. ACM, New York, NY, USA, 210–211. https://doi.org/10.1145/3125719.3132096

[37] Arjuna Sathiaseelan, Adisorn Lertsinsrubtavee, Adarsh Jagan, Prakash Baskaran, and Jon Crowcroft. 2016. Cloudrone: Micro Clouds in the Sky. In *Proc. 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet'16)*.

[38] Arjuna Sathiaseelan, Mennan Selimi, Carlos Molina, Adisorn Lertsinsrubtavee, Leandro Navarro, Felix Freitag, Fernando Ramos, and Roger Baig. 2017. Towards Decentralised Resilient Community Clouds. In *Proceedings of the 2Nd Workshop on Middleware for Edge Clouds & Cloudlets (MECC '17)*. ACM, New York, NY, USA, Article 4, 6 pages. https://doi.org/10.1145/3152360.3152363

[39] Arjuna Sathiaseelan, Liang Wang, Andrius Aucinas, Gareth Tyson, and Jon Crowcroft. 2015. SCANDEX: Service Centric Networking for Challenged Decentralised Networks. In *Proc. 2015 Workshop on Do-it-yourself Networking: an Interdisciplinary Approach (DIYNetworking '15)*.

[40] Mennan Selimi, Llorenç Cerdà-Alabern, Felix Freitag, Luís Veiga, Arjuna Sathiaseelan, and Jon Crowcroft. 2018. A Lightweight Service Placement Approach for Community Network Micro-Clouds. *Journal of Grid Computing* (28 Feb 2018). https://doi.org/10.1007/s10723-018-9437-3

[41] Mennan Selimi, Amin M. Khan, Emmanouil Dimogerontakis, Felix Freitag, and Roger Pueyo Centelles. 2015. Cloud services in the Guifi.net community network. *Computer Networks* 93, Part 2 (2015), 373 – 388. https://doi.org/10.1016/j.comnet.2015.09.007

[42] Bart Spinnewyn, Ruben Mennes, Juan Felipe Botero, and Steven Latre. 2017. Resilient application placement for geo-distributed cloud networks. *Journal of Network and Computer Applications* 85 (2017), 14 – 31. https://doi.org/10.1016/j.jnca.2016.12.015 Intelligent Systems for Heterogeneous Networks.

[43] Asser N. Tantawi. 2015. Quantitative Placement of Services in Hierarchical Clouds. In *Proceedings of the 12th International Conference on Quantitative Evaluation of Systems - Volume 9259 (QEST 2015)*. Springer-Verlag New York, Inc., New York, NY, USA, 195–210. https://doi.org/10.1007/978-3-319-22264-6_13

[44] A. N. Tantawi. 2016. Solution Biasing for Optimized Cloud Workload Placement. In *2016 IEEE International Conference on Autonomic Computing (ICAC)*. 105–110. https://doi.org/10.1109/ICAC.2016.34

[45] William Tarneberg, Amardeep Mehta, Eddie Wadbro, Johan Tordsson, Johan Eker, Maria Kihl, and Erik Elmroth. 2017. Dynamic application placement in the Mobile Cloud Network. *Future Generation Computer Systems* 70 (2017), 163 – 177. https://doi.org/10.1016/j.future.2016.06.021

[46] Davide Vega, Roger Baig, Llorenç Cerdà-Alabern, Esunly Medina, Roc Meseguer, and Leandro Navarro. 2015. A technological overview of the guifi.net community network. *Computer Networks* 93, Part 2 (2015), 260 – 278. https://doi.org/10.1016/j.comnet.2015.09.023

[47] Davide Vega, Llorenc Cerda-Alabern, Leandro Navarro, and Roc Meseguer. 2012. Topology patterns of a community network: Guifi.net. In *1st International Workshop on Community Networks and Bottom-up-Broadband (CNBuB 2012), within IEEE WiMob*. Barcelona, Spain, 612–619. https://doi.org/10.1109/WiMOB.2012.6379139

[48] Shiqiang Wang, Rahul Urgaonkar, Ting He, Kevin Chan, Murtaza Zafer, and Kin K. Leung. 2017. Dynamic Service Placement for Mobile Micro-Clouds with Predicted Future Costs. *IEEE Trans. Parallel Distrib. Syst.* 28, 4 (April 2017), 1002–1016. https://doi.org/10.1109/TPDS.2016.2604814

[49] George Xylomenos, Christopher N. Ververidis, Vasilios A. Siris andn Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. 2014. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials* 16, 2 (May 2014), 1024–1049.