PiCasso: Enabling Information-Centric Multi-tenancy at the Edge of Community Mesh Networks

Mennan Selimi[a,e], Adisorn Lertsinsrubtavee[b,d], Arjuna Sathiaseelan[c], Llorenç Cerdà-Alabern[a], Leandro Navarro[a]

[a]*Universitat Politècnica de Catalunya, BarcelonaTech, Spain*
{*mselimi, llorenc, leandro*}*@ac.upc.edu*
[b]*University of Cambridge, UK*
{*mennan.selimi, adisorn.leetsinsrubtavee*}*cl.cam.ac.uk*
[c]*Ammbr Research Labs, Cambridge, UK*
{*arjuna*}*@ammbrtech.com*
[d]*Asian Institute of Technology, Thailand*
{*adisorn*}*@ait.asia*
[e]*Max van der Stoel Institute, South East European University, North Macedonia*
{*m.selimi*}*@seeu.edu.mk*

Abstract

Edge computing is radically shaping the way Internet services are run by enabling computations to be available close to the users - thus mitigating the latency and performance challenges faced in today's Internet infrastructure. Emerging markets, rural and remote communities are further away from the cloud and edge computing has indeed become an essential panacea. Many solutions have been recently proposed to facilitate efficient service delivery in edge data centers. However, we argue that those solutions cannot fully support the operations in Community Mesh Networks (CMNs) since the network connection may be less reliable and exhibit variable performance. In this paper, we propose to leverage lightweight virtualisation, Information-Centric Networking (ICN), and service deployment algorithms to overcome these limitations. The proposal is implemented in the PiCasso system, which utilises in-network caching and name based routing of ICN, combined with our HANET (HArdware and NETwork Resources) service deployment heuristic, to optimise the forwarding path of service delivery in a network zone. We analyse the data collected from the Guifi.net Sants network zone, to develop a smart heuristic for the service deployment in that zone. Through a real deployment in Guifi.net, we show that HANET improves the response time up to 53% and 28.7% for stateless and stateful services respectively. PiCasso achieves 43% traffic reduction on service delivery in our real deployment, compared to the traditional host-centric communication. The overall effect of our ICN platform is that most content and service delivery requests can be satisfied very close to the client device, many times just one hop away, decoupling QoS from intra-network traffic and origin server load.

Key words: Information-centric networking; community networks; mesh networks; edge computing; common-pool resources.

1. Introduction

Computer networks deliver content and services to users worldwide, but these networks can develop and grow under very diverse conditions. Network infrastructures can range from carefully planned and centrally managed operator networks that can show uniform performance and balanced capacity according to demand. In the other end there are decentralized mesh networks built by citizens with little or no planning of their demand, capacity and locations, that get connected in an open and self-organized manner. These community networks, built as wireless mesh networks, defined as Community Mesh Networks (CMNs) are based on the principle of reciprocal sharing of network bandwidth. Their main purpose is both to satisfy community's demand for Internet access and to provide services of local interest.

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 the devices and links used in the infrastructure and its management. This unique characteristic makes CMNs different from the conventional ISP (Internet Service Provider) networks as the topology is dynamically changing. Hence, the current architectures and platforms in CMNs are failing to capture the dynamics of the network and therefore they fail to deliver the satisfying QoS [1] [2]. The challenges mentioned bring a lot of attention to CMNs to build infrastructures that better suited to today's use (in particular, content distribution and mobility) and more resilient to disruptions and failures.

The latest advances in lightweight virtualisation technologies (e.g., Docker, Unikernel), allows many developers to build local edge computing platform that could be used to deliver services within CMNs [3]. Despite delivering these lightweight services within a data centre is trivial, delivering them across intermittent connectivity of CMNs has a lot of challenges. 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. This approach could struggle for *service delivery* to transport the service instances to the network edge as the connectivity would fail at any time. In addition to that, those platforms do not have specific strategies on the *service deployment* in CMN environments. This raises several questions: Which services should be delivered? When should they be delivered? What are the suitable criteria for node selection to host the service? Is network-aware placement enough to deliver satisfactory performance to CMN users? However, this is not trivial and requires an effective strategy to manage the service delivery in CMNs.

On the other hand, Information-Centric Networking (ICN) has recently emerged as a potential solution for delivering named contents. The ICN leverages in-network storage for caching, multi-party communication for replication and interaction models that decouple senders and receivers. 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 origin address, where the content can be delivered from any host that currently has the content in its storage. Although ICN brings a lot of flexibilities in terms of content delivery, the current ICN implementations are rather focused on the simple static content (e.g., short message, video file). In this regard, we argue that ICN should be extended to better support transporting at the service layer.

This paper extends our previous work [4] and [5] by focusing on two main interrelated research problems: service delivery and service deployment. The former refers to the process of delivery and instantiate a service instance (e.g., web server) from the service provider to the edge computing node. The latter is the logic that decides where and when to deploy the service instance regarding the service requirements, network status and available resources (e.g., CPU load, memory). In this context, we propose *PiCasso*, a lightweight edge computing platform that combines the lightweight virtualisation technologies and a novel Information-Centric Networking (ICN) to facilitate both service delivery and service deployment in challenging environment such as CMNs. We underpin PiCasso with Docker container-based service that can be seamlessly delivered, cached and deployed at the network edge. The core of the PiCasso platform is the *decision engine* making a decision on where and when to deploy a service instance to satisfy the service requirements while considering the network status and available hardware resources. PiCasso introduces a new service abstraction layer using ICN to enable more flexibility in service delivery. Instead of hosting services in the fixed centralised location (e.g., service repository), PiCasso allows the edge devices obtaining service instance from the nearest caches by utilising inherent name-based routing and in-network caching capabilities of ICN. Furthermore, PiCasso is also integrated with a service controller and a full functional monitoring system to optimise the service deployment decision in CMNs. Specifically, our key contributions are summarized as follows:

- **First**, we characterize the performance of a real production mesh network such as Guifi.net (Guifi-Sants) and identify the bottlenecks affecting the service performance. This is achieved by monitoring the network for two months period and taking measurements regarding network status and available resources.

- **Second**, based on the measurements in the Guifi.net, we design PiCasso, a multi-access edge computing platform which deploys QoS-sensitive services at the network edge. We design the decision engine of Picasso, which selects the appropriate nodes for service instantiation based on constraints observed in our Guifi.net measurements (network bandwidth, available hardware resources and network topology) in a network zone. The HANET decision engine algorithm uses the state of the underlying network to optimize the service deployment in that zone.

- **Third**, we utilise the ICN principles in the architecture of PiCasso in order to enable more flexibility in the delivery of named data objects. To our best of knowledge this is the first ICN deployment in a production CMN such as Guifi.net.

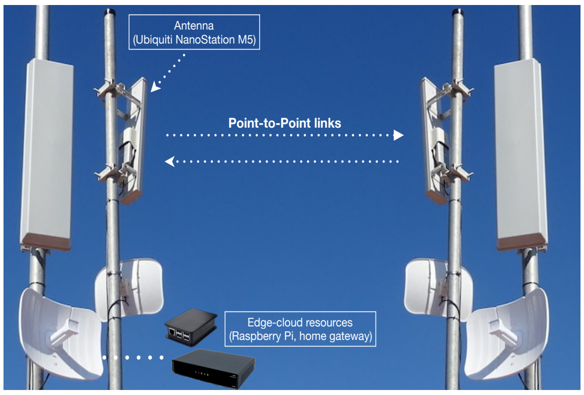


Figure 1: Outdoor Devices in Guifi-Sants Mesh Network. NanoStation M5 and Sectorial antennas are used to build point-to-point links.

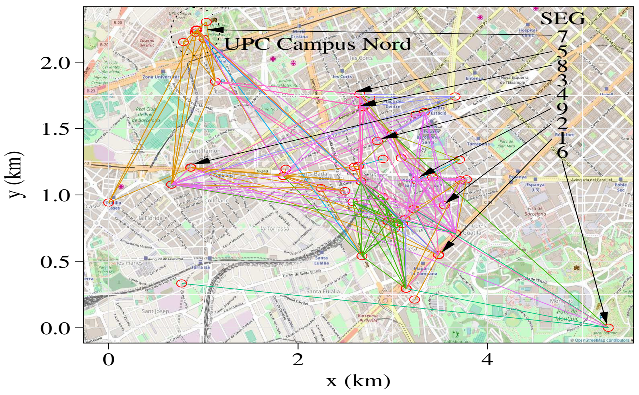


Figure 2: Guifi-Sants Network Topology (Barcelona). The location of the ORs and RPI boards in the network is shown.

The rest of the paper is organized as follows. In Section 2 we describe and characterize the performance of the Guifi-Sants mesh network. In Section 3 we present the PiCasso's architecture together with our HANET heuristic. In Section 4 the performance of the PiCasso platform is presented. Section 5 discusses our main findings. Section 6 describes related work and section 7 concludes and discusses future research directions.

2. Case Study: Guifi-Sants Mesh Network

Guifi-Sants is a subset of Guifi.net CMN, which started to operate in 2009 in a quarter of Barcelona (Catalonia, Spain) called Sants as part of the Quick Mesh Project (qMp) [1]. The main objective of the qMp project in the Guifi-Sants mesh network is to bring quick and easy WiFi networks into large, crowded events (such as concerts, demonstrations, public events, etc.) by leveraging ad-hoc and dynamic routing technologies. This is achieved by providing its own firmware for embedded network devices based on OpenWRT Linux operating system. This firmware provides an easy way to set up mesh networks - wired or wireless (WiFi) or a mix of both and allows fast and reliable way to extend an Internet up-link to end-users.

Currently (i.e., at the time of writing), the Guifi-Sants network has 80 nodes (with more than 300 active users) and continues to grow day by day. In terms of devices used, the Guifi-Sants users have an outdoor router (OR) with a Wi-fi interface on the roof as show in Figure 1. The ORs are used to build P2P (point-to-point) links in the network. They are connected through Ethernet to an indoor AP (access point) as a premises network where the edge services are running (e.g., on Raspberry Pi's, home gateways etc). The most common OR in Guifi-Sants is the NanoStation M5 (shown in Figure 1). Some strategic locations have several NanoStations, that provide larger coverage. In addition, some links of several kilometers are set up with parabolic antennas (NanoBridges). ORs in Guifi-Sants are using BMX6 as the mesh routing protocol [6] [7]. Further, in the network there are 3 gateways (proxies) distributed that connect the Guifi-Sants mesh network to the rest of Guifi.net CMN and Internet.

For our experimental cases, we deploy 10 Raspberry Pi 3 (RPI3) boards in the ORs of the network and use them as servers. Figure 2 depicts the topology of the Guifi-Sants network and the location of the RPI3 boards.

Data collection: In order to monitor the network for longer periods, we used our live monitoring system which is operational and can be seen in the following link[2]. The data is collected by connecting to each Guifi-Sants OR and running basic system commands available in the qMp distribution. This method has the advantage that no changes or additional software need to be installed in the nodes. Live measurements have been taken hourly during the entire month of September 2017 and September 2018. The collected data is publicly available in the Internet. We use this data to analyse the main aspects of the Guifi-Sants network zone.

[1]`http://qmp.cat/`
[2]`http://dsg.ac.upc.edu/qmpsu/index.php`

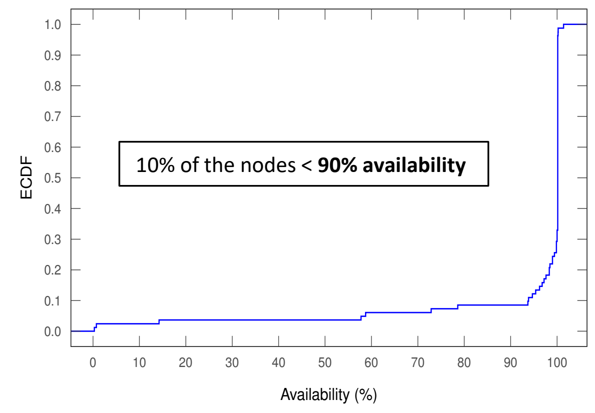


Figure 3: Availability of the nodes (Guifi-Sants network)

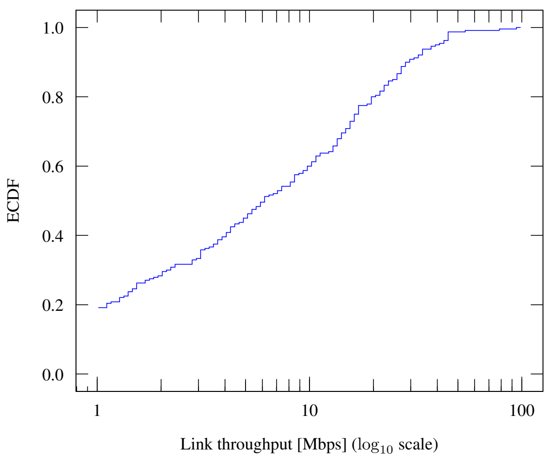


Figure 4: Link bandwidth distribution (Guifi-Sants network)

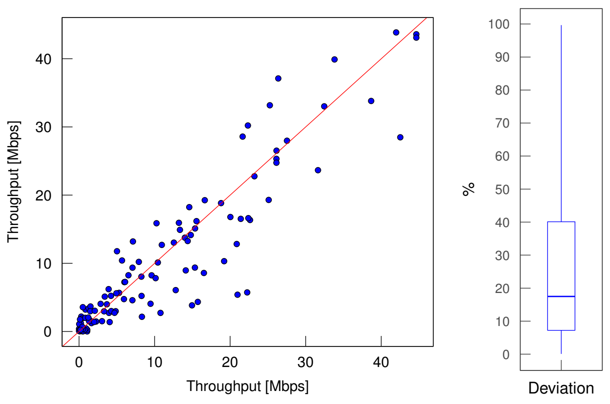


Figure 5: Link bandwidth asymmetry (Guifi-Sants network)

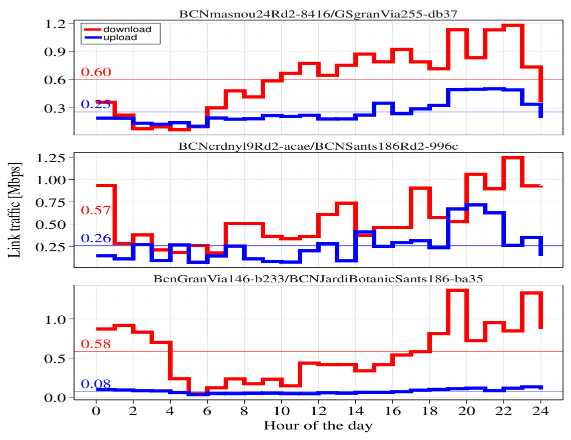


Figure 6: Traffic in 3 busiest links (Guifi-Sants network)

2.1. Guifi-Sants Network Characterization

Network and user-focused services: One possible reason for the absence of the local community-owned services in Guifi-Sants and Guifi.net was because of the difficulty to deploy such services and the shortage or individuals, companies and organizations interested in the commercial operation of these services. To overcome these issues, one solution for CMN enthusiasts was to design community network micro-cloud distributions such as Cloudy [8], Guinux [9] etc., where users were able to deploy their preferred services and share with the others in community networks. Micro-clouds were open user-driven clouds formed by community-managed computing resources which were able to run different open source distributions (e.g., operating systems) installed by the users. Key characteristics of these distributions were a set of scripts that automated the configuration process of services. However, there was not a logic behind the service deployment. The service deployment heuristics were agnostic to the state of the underlying network and this could lead to important inefficiencies in terms of latency, performance, cost, availability etc.

There are two type of services in the network: *network-focused* (network graph-servers that collect network monitoring data and provide graphs, DNS Servers, NTP servers etc) and *user-focused* services (proxy servers, VoIP, video, instant messaging etc). Considering that network management is of interest to all users in the network (i.e., to keep the network up and running), services related to the network operation outnumber the local services intended for end-users [2]. Moreover, the most frequent of all the services offered, whether user-focused or network-focused, are the proxy services [10]. Proxies act as free gateways to the Internet for the community network users.

Network Topology: The network topology of the Guifi-Sants networks and Guifi.net CMN in general, is different with respect to conventional Internet Service Provider (ISP) networks. Guifi.net is composed of numerous distributed CMNs and they represent different types of network topologies. The overall topology can change and there is no fixed topology as in the data center (DC) environment. The network has a mesh topology in the backbone, and each node of the backbone (i.e., super-node) provides access to the client nodes [11]. The backbone interconnects different super-nodes by point to point links, local networks provide access to the client nodes. Figure 2 depicts the topology of the Guifi-Sants network in Barcelona. Guifi-Sants network shows some typical patterns from the urban networks (i.e., mesh networks) combined with an unusual deployment, that does not completely fit either with organically grown networks or with planned networks. 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.

Node and Link Characteristics: Figure 3 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*, counted since the node shows up for the first time. A capture is an hourly network snapshot that we take from the Guifi-Sants network (i.e., we took 744 captures in total). Figure 3 reveals that 10% of the nodes have availability lower than 90% and others nodes left have availability between $90-100\%$. In a mesh network such as Guifi-Sants, 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.

Figure 4 depicts the average bandwidth distribution of all the links in the network. The average bandwidth observed in the network is 11.7 Mbps. Further, Figure 4 reveals that the 60% of the nodes have 10 Mbps or less throughput. The average bandwidth of 11.7 Mbps obtained in the network can be attributed to the 802.11a/n devices used in the network. In order to measure the link asymmetry, Figure 5 shows the bandwidth measured in each direction of the link. A boxplot of the absolute value of the deviation over the mean bandwidth of every link in both directions is also depicted on the right. The figure shows that around 25% of the links have a deviation higher than 40%. At the same time, only 25% of the links have a deviation lower than 10%. After performing some measurements regarding the signaling power of the devices, we discovered that the main reason of this significant asymmetry is that some of the community members have re-tuned the radios of their devices (e.g., transmission power, channel and other parameters), trying to achieve better performance, thus, changing the characteristics of the links.

Figure 6 shows the average traffic of three representative end nodes. The measurements correspond to the links used by three users to access the mesh. Averages have been taken in both directions (download and upload) over each hour of the day. As expected, Figure 6 shows a strong asymmetry between download and upload traffic: the download daily average is around 0.6 Mbps for all users, while the upload is only around 0.25 Mbps or less. The figure also shows a significant difference of traffic during the day, reaching in all cases more than 1.2 Mbps in the busiest hour.

2.2. Key Observations

Here are some observations that we have derived from the measurements in the Guifi-Sants mesh network:

A lack of smart service platforms: Despite achieving the sharing of bandwidth, Guifi-Sants and Guifi.net in general, have not been able to widely extend the sharing of local alternatives to popular cloud services, such as private data storage and backup, instant messaging, media sharing, social networks 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 network micro-clouds [8] but without major adoption. Further, a growing number of micro-cloud services desire computational tasks to be located nearby users. They include needs for lower latency, a better-user experience and efficient use of network bandwidth. *Currently, there is no open source platform to bootstrap and manage decentralized community services. Open and self-managing solutions with smart decision making algorithms can definitely boost the adoption of local services in the network.*

Variability in topology and change in capacity load: The Guifi-Sants network is highly dynamic and diverse due to many reasons, e.g., its community nature in an urban area; its decentralized organic growth with extensive diversity in the technological choices for hardware, wireless media, link protocols, channels, routing protocols etc.; its mesh topology etc. The current network deployment model is based on geographic singularities rather than QoS. The network is not scale-free. The topology is organic and different with respect to conventional ISP networks. This implies that a solution (i.e., algorithm) that works in a certain topology might not work in another one. *There is a need* *for a fast, adaptive and effective heuristics that are not agnostic to the state of the underlying network, and heuristics that can cope with the topology dynamics.*

Non-uniform resource distribution: The resources are not uniformly distributed in the network. Wireless links are with asymmetric quality for services (25% of the links have a deviation higher than 40%). There is a highly skewed bandwidth, traffic and latency distribution. The symmetry of the links, an assumption often used in the literature of wireless mesh networks, is not very realistic for our case and algorithms (heuristics) unquestionably need to take this into account. *Currently, services in the Guifi-Sants and Guifi.net in general, are placed in specific servers where operators have access. This manual and centralized model is quite ineffective, failing to capture the dynamics of the network, take into consideration the location of clients, and therefore it can easily fail to deliver a satisfying QoS for most users.*

From above observations, we define the edge computing for community networks as a capability to move the operating services in the cloud to the edge of the network while considering the resource constraint environment. Optimisation can be achieved on the basis of the universality of the devices and lightweight virtualisation technologies. Universality allows them to host any kind of service whereas lightweight virtualisation allows quick and seamless deployment, re-allocation, replication and aggregation of lightweight virtual machines. The implementation of resource constrained clouds at the edge, raises several limitations. For instance, services running on the edge devices are supposed to be less complex and static such as web service or lightweight video streaming server, the centralised cloud computing with Internet connect is still needed . However, those simple services are commonly used in the community networks as already mentioned in previous subsection. From these insights, we design PiCasso, a low-cost edge computing platform that operates at the ”extreme” edge of community network. We present a system architecture and fully implement it on a real hardware (discussed in Section 3). We showcase the efficiency and effectiveness of PiCasso by focusing on its core features. First, we show how PiCasso achieves a better end-user experience (e.g., low latency, great responsiveness) using the HANET heuristic (discussed in Section 3.3.2) in a network zone. Then, we show how PiCasso achieves more efficient use of network bandwidth at low network cost in Guifi-Sants network using its ICN capabilities (discussed in Section 4.2).

3. PiCasso: Multi-Access Lightweight Edge Computing Platform

PiCasso[3] is implemented based on the service and access abstraction where lightweight virtualisation services are delivered through the Information Centric Networking (ICN). There are several ICN implementations [12, 13, 14, 15, 16] that have been proposed during the past decade. Among those implementations, Named Data Networking[4] (NDN) is the most suitable candidate for PiCasso as it uses a simple stateful forwarding plane to utilise the distributed in-network caching without any control entity. Thus, we developed PiCasso that extends NDN protocol stack to support service delivery and service deployment in CMNs.

3.1. Platform Overview

PiCasso is a lightweight edge platform that can rapidly deliver services to end users at the edge, in a given network zone, even though the network connectivity is intermittent. PiCasso relies on service and access abstraction where lightweight virtualisation services are delivered through the ICN. This approach that does not rely on underlying host-centric networking model, decouples the service from it’s physical location by taking advantage of the naming and content caching that could be used to make intelligent forwarding decisions and publish/subscribe communication primitives that allow asynchronous communications etc.

The overview of PiCasso platform is presented in Figure 7. The key entity of PiCasso is referred to *Service Controller* (SC) that periodically observes (i.e., monitors) the network topology and resource consumption of potential nodes for the service deployment in a given network area. As typical in edge networks, that can include both service consumers and providers, usually called a network zone, related to the more general concept of an autonomous system. A service offered in multiple zones would involve a SC that can optimize the service in each zone, therefore service control of a global service will follow a hierarchical structure. Therefore, from now we will focus on one of these edge

[3]`https://github.com/AdL1398/PiCasso.git`
[4]`https://named-data.net/`

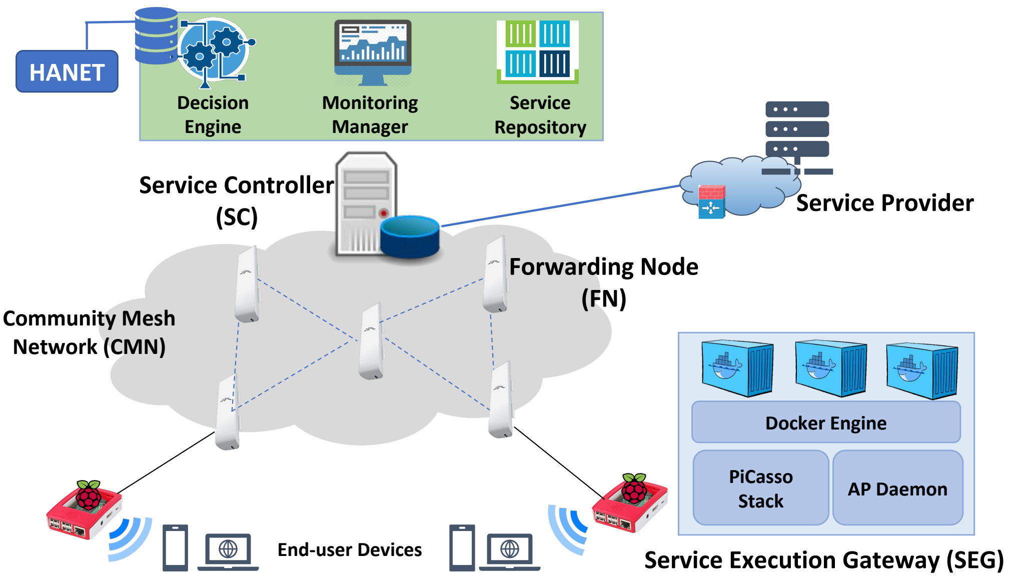


Figure 7: The overview of the PiCasso platform. Main components of Picasso platforms are shown: Service Controller (SC), Service Execution Gateway (SEG) and Forwarding Node (FN).

regions. 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 maintain a good QoS and overcome the network connectivity problems, SC augments the monitoring data along with service deployment algorithms to decide where and when to place the services. The *Service Execution Gateway* (SEG) provides a virtualisation capability to run a service instance at the network edge (e.g., users home). 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[5]) 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[6] on ARM Cortex-A53 1.2 GHz 4 core CPU with 1 GB RAM. The *Forwarding Node (FN)* in Figure 7 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 necessary need to execute the services.

3.2. Architecture of PiCasso

Current implementations of edge computing still rely on a centralized approach with large amount of traffic spreaded out over the network. Streams of requests are sent from edge devices to the data center instead of fetching the contents and services from the nearest nodes. NDN (Named Data Networking) is one of the ICN projects which instead of using IP address for communication, NDN directly addresses the contents by name regardless of physical location. As a matter of fact in NDN, a piece of content or service can be stored or cached at multiple locations. NDN uses name based forwarding (NFD) where the routing should be done dynamically and effectively to fetch the desired contents of services from the best location. NDN naturally fits with the nature of CMNs allowing nodes located far away and have intermittent connectivity to retrieve the content or services directly from the nearest caches.

Currently, PiCasso is written in Python and implemented on top of NDN protocol stack [12] and Docker technology [17]. The main function blocks of PiCasso's architecture are presented in Figure 8 and are the following ones:

[5]`https://wiki.gentoo.org/wiki/Hostapd`
[6]`https://blog.hypriot.com/downloads/`

- **NFD Forwarding plane** sits between application and transport layer 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. This NFD layer inherits the functionalities from the original NDN code base which maintains three types of data structure: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). The FIB operates as a named based routing table. Unlike the traditional IP routing table, FIB records the name prefix with outgoing interface rather than IP or network prefixes. The PIT keeps track the pending Interest message that have not yet been delivered with the matching content. In PIT table records the incoming faces along with Interest name while waiting for a response of matching name content. CS is a local cache integrated in every NDN nodes. The NFD processes a content request (Interest message) by interrogating the matching data with name prefix. If data exists in the CS, the NFD returns the Data message to the same interface from which Interest message arrived. Otherwise the NFD looks up the name prefix in the PIT, and if a matching PIT entry is found, the NFD adds a new entry of newly arrived Interest with an incoming interface and discards the Interest without forwarding. In case neither CS nor PIT find a matching record, the NFD further forwards the Interest message regarding information in the FIB.

  In PiCasso, we have also extended the NDN protocol stack by introducing a DTN (Delay Tolerant Networking) face to facilitate operation in challenge network environment like post-disaster scenario. Notice in NDN implementation, "face" is used instead of "interface". The face refers to the logical interface. For instance, a physical interface (e.g., WiFi) can have multiple faces with different name prefixes. This new DTN 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 can be found in [18]. We integrate the NFD forwarding plane to PiCasso architecture through a Python wrapper of NDN APIs, called PyNDN[7].

- **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[8], a Python wrapper for Docker to expose the controlling messages to Docker engine.

- **Monitoring Agent** is responsible for measuring the current status of underlying hardware resources (RPI3 devices and ORs) such as current memory usage, CPU utilisation, CPU load, network bandwidth, RTT, availability etc, and reporting this data to SC. Further, it associates with Docker engine to report the status of running containers (e.g., container names, number of running containers) and resource consumption inside each container (e.g., CPU and memory usage).

- **Decision Engine (DE)** or orchestrator is responsible for selecting an appropriate SEG node for service instantiation based on constraints such as available hardware resources, QoS and network topology. DE has access to algorithm repository that can execute to make decisions on deployment of service instances. The service deployment algorithms can be dynamically updated regarding different deployment scenarios and service requirements.

- **Service Repository** is a repository for storing dockerized compressed service images. Images of the services are stored augmented with specification about service deployment in the form of JSON format. Our implementation allows the third party service providers to upload their service along with a deployment description augmented with specifications and QoS requirements.

- **Monitoring Manager** periodically collects the monitoring data from each SEG and stores in the database (Monitoring DB). It is implemented based on a time series database, called InfluxDB[9]. We also implemented the dashboard for monitoring system using Grafana[10] to visualise time series data for SEG's measurements and application analytics. Figure 9 shows the user interface of PiCasso monitoring dashboard.

[7] `https://github.com/named-data/PyNDN2`

[8] `https://github.com/docker/docker-py`

[9] `https://github.com/influxdata/influxdb-python`

[10] `https://grafana.com/`

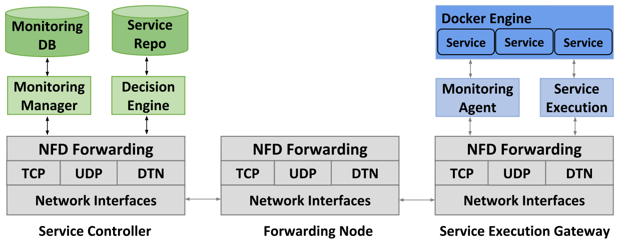


Figure 8: PiCasso's function blocks

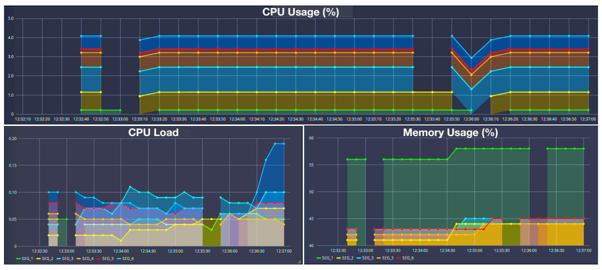
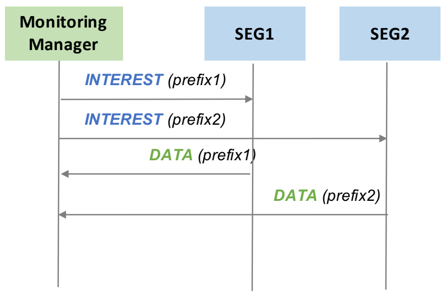
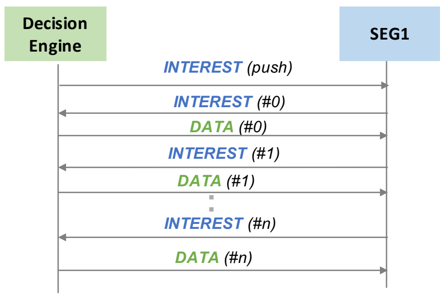


Figure 9: PiCasso monitoring dashboard



(a) Pull-based model



(b) Push-based model

Figure 10: Key operations in PiCasso. *(a)* Monitoring manager retrieves the monitoring data from SEGs. *(b)* Decision Engine delivers the service to the SEG

3.3. Operations in PiCasso

In this section we explain the main operations performed by PiCasso as well as the benefits of using NDN stack for content and service delivery.

3.3.1. Collecting monitoring data

The monitoring manager periodically places the pull requests against the monitoring agent of each SEG to collect the current status of their resources. This operation follows the native pull-based communication model of NDN. As shown in Figure 10a, 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 SEGs receives this pull Interest message, they attach the current monitoring data with JSON format to the Data message and forward to the same path that Interest message (reverse path forwarding) came from by using information in the PIT (Pending Interest Table). To avoid receiving outdated data from the caches, we set the data freshness to a small value (e.g., 10 ms).

3.3.2. Service deployment heuristic

The Decision Engine (DE) of PiCasso relies on the service deployment algorithms to decide where to place services in a given network zone aiming to maximise the QoS by optimizing the usage of scarce resources in CMN such as bandwidth. The DE 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 for deploying and then instantiating the service.

In this work, we propose HANET (HArdware and NETwork resource) heuristic algorithm, which is designed specifically for service deployment in unreliable network environment such as wireless CMN zone. HANET uses the state of the underlying CMN (i.e., the Guifi-Sants zone) to optimize service deployment. In particular, it considers three sources of information: i) network bandwidth and ii) node availability and iii) hardware resources to make optimized decisions. Note that other sources of information could be also considered, e.g. demand/load. For the sake of simplify we have assumed light demanded services, dominated by the aforementioned information. First, we test HANET with the static data obtained from the Guifi-Sants network (i.e., bandwidth, availability and CPU data). Then we ran HANET in Guifi-Sants zone and quantify the performance achieved after deploying real services (discussed in Section 4). The HANET heuristic algorithm (see Algorithm 1) runs in three phases:

Phase 1 - Setting up the network: In this phase, based on the raw data that we got from the Guifi-Sants zone, initially we build the topology graph of the network. The topology graph of the Guifi-Sants network zone is constructed by considering only operational nodes (i.e., working) and having one or more links pointing to another node (i.e., we remove the disconnected nodes). Once the topology graph is built, 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 k clusters of nodes based on the Euclidean distances between them, where the distance metrics in our case are the geographical coordinates of the nodes, a simple and good enough coordinate and Euclidean distance for K-Means calculation, expressing preference for closer nodes, reflecting lower number of hops and better performance. 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.

Phase 2 - Bandwidth Computation/Estimation: The second phase of the algorithm is based on the concept of finding the cluster *heads* maximizing the bandwidth 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 cluster heads computed are the candidate nodes for the service placement.

Phase 3 - Content Placement: The third phase is executed after the cluster heads are computed in Phase 2. Based on that, the services are placed on the selected cluster heads if their CPU load is under the predefined threshold. This corresponds to pulling the service images from the service repository, pushing at the selected edge nodes (cluster heads) and starting them.

HANET Performance and Complexity: Figure 11 shows the average bandwidth from the cluster heads (i.e., obtained with different heuristics) to the other nodes of clusters. We compare HANET heuristic performance with *random* (default strategy in Guifi.net and Guifi-Sants) and *K-Means (Phase 1 of the algorithm)*. Figure 11 reveals that for the considered number of services k, HANET outperforms both *K-Means* and *random* placement. For $k = 2$, the average bandwidth to the cluster heads has increased from 18.3 Mbps (*K-Means*) to 27.7 Mbps (HANET), which represents a 33.8% improvement. The highest increase of 45.67% is achieved when $k = 11$. Based on the observations from Figure 11, the gap between the two algorithms grows as k increases. We observe that k will increase as the network grows. And hence, HANET will presumably render better results for larger network zones than the rest of strategies.

The complexity of the *HANET* is as follows: for *HANET*, finding the optimal solution to the K-means (i.e., phase one) clustering problem if k and d (the dimension) are fixed (e.g., in our case $n = 77$, and $d = 2$), the problem can be exactly solved in time $\mathcal{O}(n^{dk+1} \log n)$, where n is the number of entities to be clustered. The complexity for computing the cluster heads in phase two is $\mathcal{O}(n^2)$, and $\mathcal{O}(n)$ for the reassigning the clusters in phase three. Therefore, the overall complexity of HANET is polylogarithmic $\mathcal{O}(n^{2k+1} \log n)$, which is significantly smaller than the brute force method and thus practical for commodity processors.

3.3.3. Delivering services to the edge

In PiCasso, the service images are kept in service repository which is the repository for providers to upload their services. To distribute the services over the network, PiCasso requires the push-based communication model to push the service from the service repository to the SEG at the edge of the network. However, PiCasso relies on NDN which supports only pull-based model where a consumer (or receiver) has to initiate the communication. To support this operation, we have implemented the push-based communication model based on Interest/Data exchange of primitive NDN. We follow the publish-subscribe model [19] 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 10b 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. As in NDN, a content is divided into several chunks, the last prefix is

Algorithm 1 HANET (Hardware and Network-aware Service Placement) Heuristic

```
Require: topo = GuifiSantsTopology.xml
    λ                                          ▷ availability threshold
    α                                          ▷ CPU threshold
    Rn                                         ▷ availability of the node n
    CPUch                                      ▷ CPU load of the cluster head
    Bi                                         ▷ Bandwidth of the node i

    Phase 1 – Setting up the network zone:

 1: procedure NetworkSetup(topo)
 2:     g = BuildTopology(topo)
 3:     $g' = SanitizeGraph(g)$
 4:     for each item in $g'$ do
 5:         Remove disconnected nodes
 6:         Ensure bidirectional links
 7:         Remove nodes with no metrics
 8:     end for
 9:     if $Rn \geq \lambda$ then
10:         PerformKMeans(g', k)
11:         return C
12:     end if
13: end procedure

    Phase 2 – Bandwidth Computation/Estimation

14: procedure ComputeHeads(C)
15:     clusterHeads ← list()
16:     for all $k \in C$ do
17:         for all $i \in Ck$ do
18:             Bi ← 0
19:             for all $j \in setdiff(C, i)$ do
20:                 Bi ← Bi + estimate.route.bandwidth(g', i, j)
21:             end for
22:             clusterHeads ← $\arg\max_C \sum_{i=1}^{k} \sum_{j \in Ci} B_{ij}$
23:         end for
24:     end for
25:     return clusterHeads
26: end procedure

    Phase 3 – Content Placement (Hardware)

27: procedure PlacementPhase
28:     for each clusterHeads do
29:         if $CPUch \leq \alpha$ then
30:             DeployService()
31:             StartService()
32:         end if
33:     end for
34: end procedure
```

reserved for the requesting chunk ID which is started from zero (e.g., #00).

PiCasso relies on NDN stack in order to benefit from dynamic in-network caching and name-based routing capa-

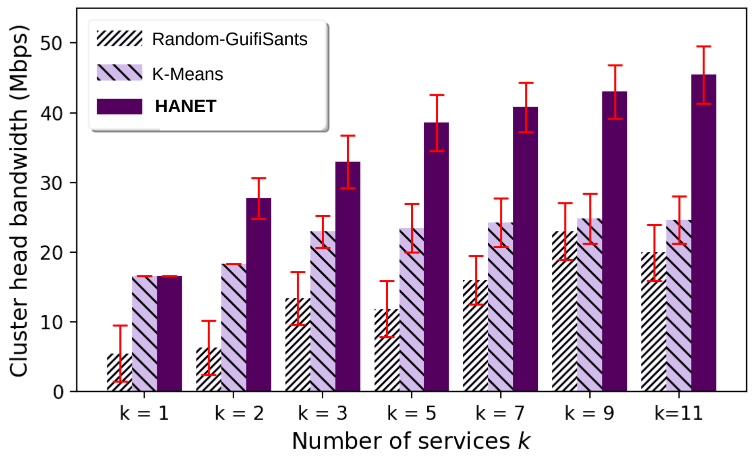


Figure 11: Average bandwidth to the cluster heads (candidate nodes for service placement)

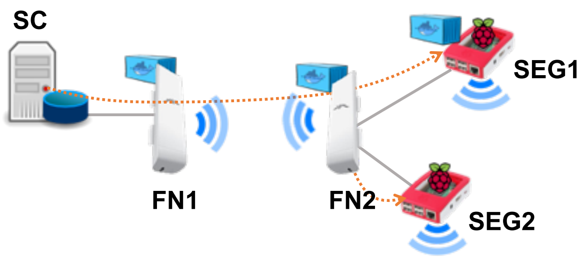


Figure 12: *PiCasso benefits from NDN mechanism as SEG2 can retrieve the service from the cache*

bilities. Figure 12 shows an example that highlights this efficiency by considering a scenario where the DE decides to deliver the service to SEG1 and SEG2. Note that the DE is running inside the service controller. At first, the DE initially sends the push Interest message to SEG1 as shown in Figure 10b. During the service delivery process, the forwarding nodes along the path naturally store content chunks in their cache (CS). As for the second stage, the DE subsequently sends the push Interest with the same service name to SEG2. Thanks to the named based routing of NDN and caches by the former request, SEG2 can opportunistically fetch the content from the nearest forwarding node (e.g., FN2) without unnecessary route towards the DE. This is very helpful for the service delivery in community network where connectivity is not always stable.

3.4. PiCasso in Guifi-Sants

In order to understand the feasibility of running PiCasso and to ensure the possible gains of our network-aware service placement heuristic HANET in a real production CMN, we deploy PiCasso in small servers connected to the nodes of the Guifi-Sants network zone, located in the city of Barcelona. We have strategically deployed 10 SEGs to cover the area of Guifi-Sants network as presented in Figure 2. In our configuration, SEGs are connected to the ORs via Ethernet cable and the service controller is centrally set up inside the main campus of Universitat Politecnica de Catalunya (UPC) where the Guifi research lab is located. We consider lightweight services based on Docker container which can be easily deployed at the edge (i.e. SEG node). The example of these services include the Video streaming where the streamer is running inside the web server container (e.g., tomcat, apache). Notice that web services like GStreamer are commonly used in Guifi-Sants. Another example refers to IoT related service that requires basic data processing right at the edge. This includes collecting the sensor data as well as simple calculation like finding the average/max/min value of sensors (e.g., temperature, humidity, CO, CO2).

Node Location: The location of five SEGs deployed is chosen based on the output of the HANET algorithm. Initially, using data from the ORs, HANET outputs the location of five ORs. Based on this, we deploy five Raspberry

Pi's to the selected ORs given by the HANET algorithm. This corresponds to the top-ranked nodes (i.e., cluster heads) selected from the HANET; with higher bandwidth, availability and CPU resources. Then, using the data coming from the RPI devices HANET is executed to find the best node (RPI device) for service deployment among five selected in the earlier phase. The other five ORs in the Guifi-Sants 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 i.e., nodes with high betweenness centrality etc).

ICN Overlay: Our deployment in Guifi-Sants follows the ICN-as-an-Overlay approach [20] by constructing an ICN shim layer on top of the existing routing protocol (i.e., BMX6). The name based routing in ICN layer is maintained by the NFD forwarding plane. In this trail, 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 ORs in the Guifi-Sants network.

Network Performance: In order to understand the possible effect of the network (e.g., background traffic) when running our experiments, we measure the bandwidth and RTT (Round-trip time) of the Guifi-Sants network. Figure 13 shows the RTT and the bandwidth of the SEGs during the time when we run experiments in the Guifi-Sants network. For each SEG (in total 10) we plot the mean throughput and the mean RTT to every other SEG (for each SEG we plot 9 values) in the boxplot form. The values are sorted from lower to higher throughput median. We can observe that, the lower the throughput is the higher the RTT is. Some of the SEGs in the network are connected with high-bandwidth links reaching a bandwidth of 30 Mbps and others with really low bandwidth links i.e, 2 Mbps.

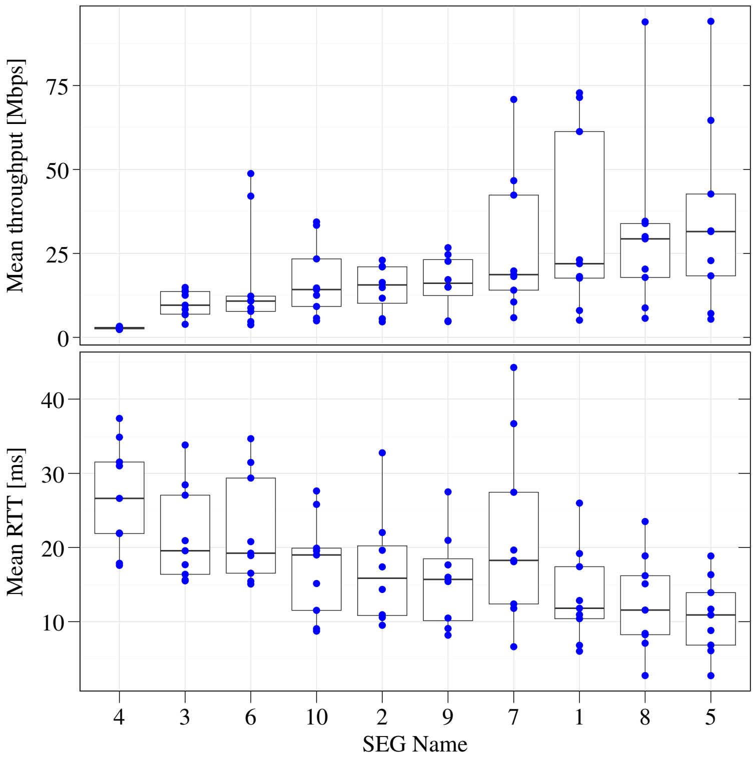


Figure 13: RTT and bandwidth of the deployed SEGs

4. Performance Evaluation

In this section we demonstrate the performance of PiCasso platform in a zone of a production mesh network such as Guifi-Sants. We concentrate on benchmarking two services: user and network-focused services. From the user services, we quantify the performance of the HANET heuristic using a stateless service (ApacheBench [21]) and a stateful Web2.0 service (Cloudsuite Web Serving benchmark [22]). The evaluation of end-user services is based on the web technology while measuring the response time is the key performance metric. The evaluation of network services focuses on the efficiency of service delivery in PiCasso comparing to a traditional host-centric communication (HCN) approach.

4.1. Evaluation of End-user Services

One of the major goals of PiCasso is to improve the QoS by delivering services close to the users at the network edge. This can alleviate the impact of intermittent connectivity problem as well as reducing the latency to access a service. Deploying multiple service instances can significantly improve the QoS in a network zone, since servers or containers can balance the load and improve response to user requests. However, in practice, it is not trivial to have 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, the HANET service deployment heuristic allows to decide where (in which nodes) to place the services. We compare the HANET heuristic with the random heuristic i.e., the usual strategy in the Guifi-Sants network zone[11].

4.1.1. Stateless user services

In the evaluation with Apache tool, we focus on the response time of the HTTP requests while considering different number of service replicas (e.g., $k = 1$ and $k = 2$). The location of the service replicas is determined by the HANET algorithm. Based on HANET, $\{SEG1\}$ and $\{SEG1, SEG8\}$ are selected for $k = 1$ and $k = 2$ respectively, as highlighted in Figure 2. 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 10b. In each node, we configured the Apache tool to simulate 10 clients instructed to send 500 HTTP sequential requests to the closest replica.

Figure 14 depicts the CDF of response time collected from the Apache client nodes. Generally, HANET achieves *significantly* lower response times compared to a random choice. 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 approach, respectively. Further, 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 down to less than 190 ms while a random choice results in 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 a static web application.

4.1.2. Stateful user services

From the stateful user services we are experimenting with the Web 2.0 service which mimics a social networking application (e.g., Facebook). For our experiments, we use the dockerized version of the CloudSuite Web Serving benchmark [22]. 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 the Elgg [23] social networking engine and it connects to the memcached server and the database server. The clients (implemented using the Faban workload generator) send requests to login into the social network and perform different operations.

We use 10 SEGs attached to the Guifi-Sants mesh routers, where 9 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 15 depicts three Cloudsuite operations performed when placing the web server with the HANET and random heuristic. Figure 15 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

[11]As the criteria for server choice usually depends on which server the service provider has an account, since they know and convince the owner, and not according, therefore randomly, to location and characteristics of service demand.

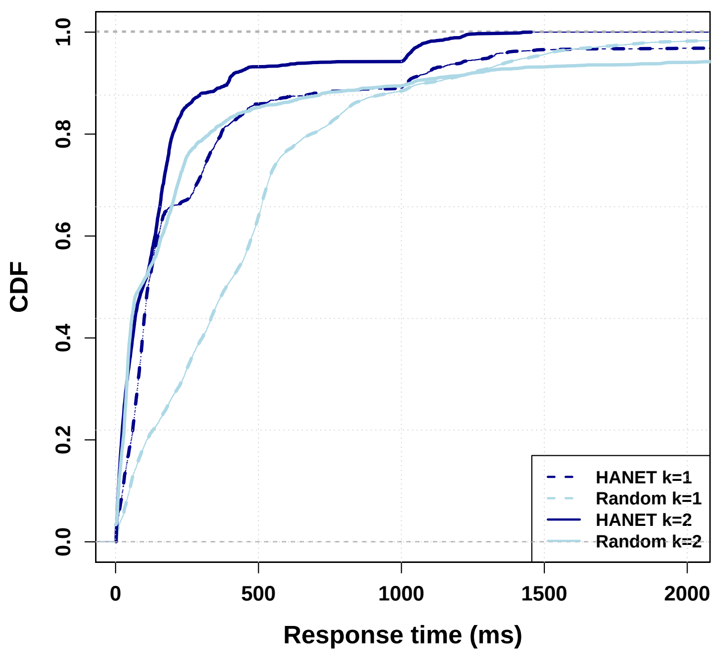


Figure 14: Response time of HTTP requests

average 53% improvement when performing 40 operations per client). Further, Figure 15 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 can have higher response times.

4.2. Evaluation of Network Services

In order to evaluate the network services, we focus on service delivery capability of PiCasso by 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 [8] and Paradrop [24]. To implement this approach, we disable in-network caching facility of PiCasso and direct the service to be delivered from the service repository to each SEG, which is also similar to the IP unicast.

4.2.1. Analysis of service delivery cost

We select four dockerised containers which have different image sizes from the Docker hub and migrate them from the service repository to all deployed SEGs. Table 1 shows a comparison between HCN and PiCAsso-HANET solution in terms of average delivery time with different size Docker images.

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 16. By comparing HCN and PiCasso, we observe that every SEG is improving the performance 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 17).

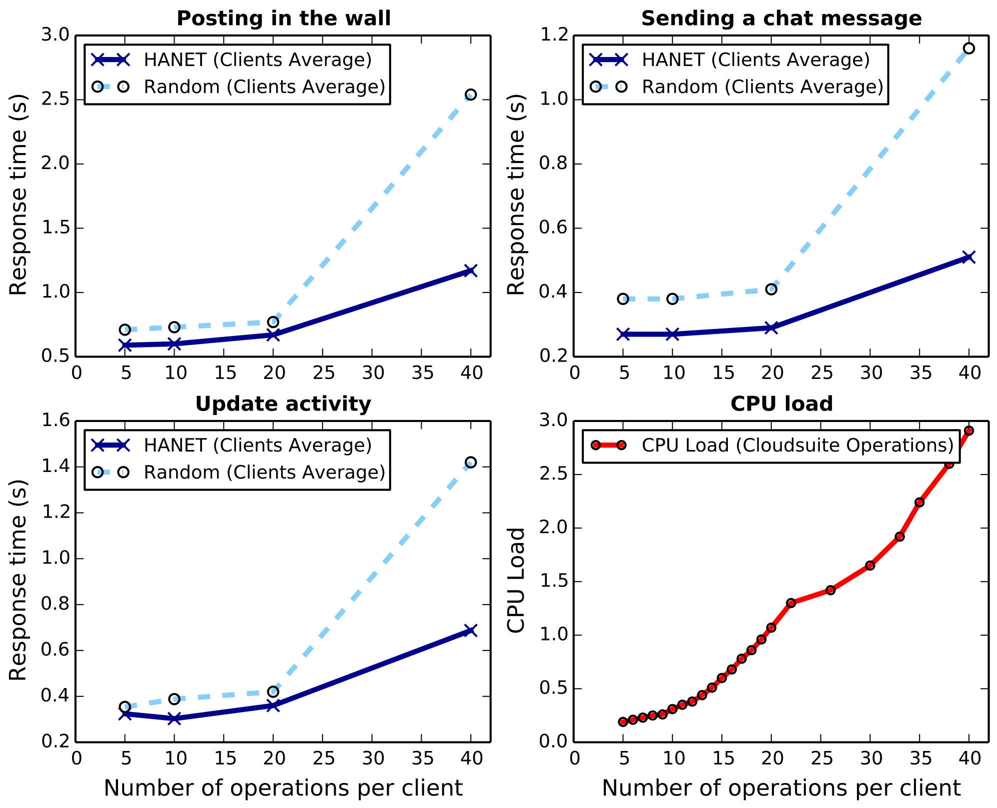


Figure 15: Cloudsuite Operations (HANET vs. Random)

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

Table 1: Comparison of the average delivery cost

On the other hand, the HCN approach is inefficient in terms of bandwidth utilisation. The red line in Figure 16 shows the link bandwidth with 95% confidence interval between each SEG and service repo using iperf (TCP traffic). Given an example of SEG2, the average link bandwidth is 24.63 Mbit/s. However, the HCN approach acquires 186.3 second to deliver the service which is approximately 6.22 Mbit/s (image size of 145 MBytes). As previously stated in Section 2.1, the resources in Guifi-Sants network are not uniformly distributed. This indicates that the traditional HCN approach is not sufficient to support good quality service delivery in this dynamic environment. In contrast, PiCasso significantly improves the bandwidth utilisation, for instance the delivery cost of SEG2 is only 50.46 second which is equivalent to bandwidth of 22.99 Mbit/s.

4.2.2. Investigating traffic consumption of service delivery

Previous results demonstrated that PiCasso efficiently improves the service delivery in Guifi-Sants 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 [25]. However, the information from these reports contains only the traffic of an overlay network. To construct the actual traffic that spread over the Guifi-Sants network, we map the paths from PiCasso overlay with the

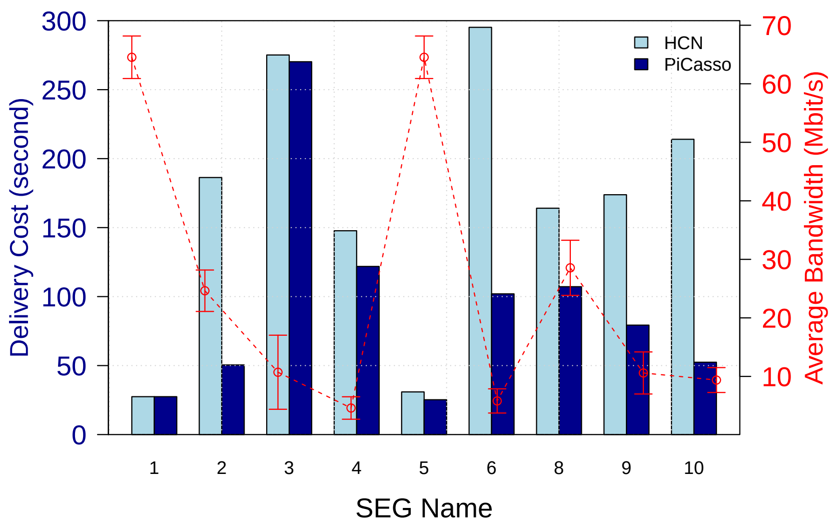


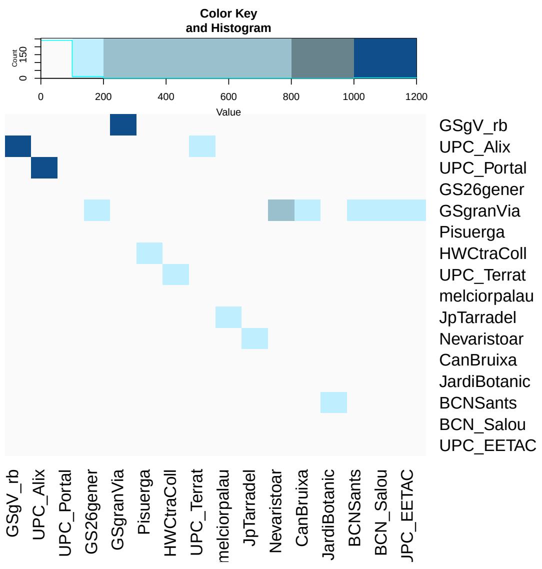
Figure 16: Inspecting the delivery cost of each SEG

routing tables of BMX6 routing protocol. For instance, the path between service controller and SEG5 (see Figure 2) can be mapped to *UPC-Portal - UPC-Alix - GSgV_rb - GSgranVia - CanBruixa* (i.e., names denote the location of routers).

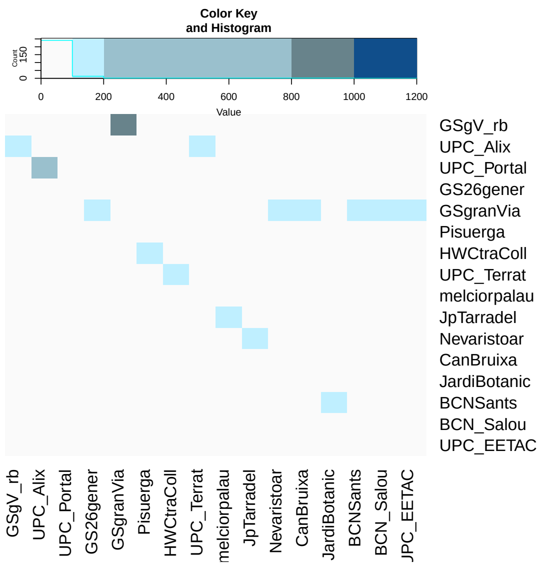
Figure 17 depicts the distribution of data traffic transferred among mesh routers to deliver a service image to all 10 SEGs. Here, we solely present the results of delivering *armbuild/debian* image (the largest image size in the experiments) due to space constraints. The total amount of traffic consumed 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 Guifi-Sants 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 to retrieve a service image from closer nodes. As illustrated in Figure 2, 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 utilised closer to the network edge. In addition to traffic reduction, PiCasso also improves the service delivery process with lower cost when deploying service instances.

5. Discussion

Local Service Ecosystem: The PiCasso edge computing platform, combines a set of NDN tools that simplify and optimize the delivery of content and services to clients, a kind of local CDN, ideally with presence of PiCasso support in the first hop, the access point. The result is that the indirection infrastructure offloads a majority of requests, decoupling content and service from the volume of demand. This encourages CMN users to participate as active contributors of services, ultimately creating an ecosystem of local services, as PiCasso can automate the load spreading across several servers and nearby network links, and increase the robustness of service delivery. PiCasso packages together different cloud services and content at near minimal network and server cost to end users. However, the challenge for Picasso remains to analyze and optimize the delivery of different kinds of services when using the ICN paradigm. For instance, one of the services to consider in our future work is live video streaming, that can benefit from taking advantage of the application semantics.



(a) Experiments with HCN



(b) Experiments with PiCasso

Figure 17: Data traffic distributed over the Guifi-Sants 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.

Deployment benefits (transparency): The Picasso platform is easy to deploy thanks to the plug-and-play feature of nodes. The adoption of the Picasso platform requires minimal changes in the WISP (Wireless Internet Service Provider) architecture or network configuration since nodes are added via plug-and-play. Moreover, PiCasso nodes are able to discover the closest node and dynamically retrieve the service image from the nearest cache. Hence, content can be transparently delivered, cached and deployed at the network edge, at just one network hop from the client.

Traffic reduction benefits (Operator gain): As mentioned in our analysis, network bandwidth is crucial in Guifi-Sants network since it highly fluctuates. Even though the end-user services could not gain much benefits from ICN capability as we haven't intervene the end user to upgrade their devices to support the ICN. In contrast, only the network equipments (e.g., SEG node) in Guifi.net are ICN ready. The use of ICN over PiCasso platform results in significant traffic reduction in network zones from the benefits of in-network caching and name-based routing. These functions assists PiCasso to reduce the service delivery cost as well as the network traffic during the service deployment (42% reduction in terms of traffic comparing to a host-centric solution). However, regarding the results from our deployment trial, the traffic consumption of PiCasso is not yet optimized. As a matter of fact, NDN strictly requires the collaborative effort in order to achieve the maximum bandwidth reduction from in-network caching capability. However, in our case, we couldn't deploy the PiCasso node (SEGs) in all locations. For instance, the owner of GSGgranVia node, one of the most critical node from our selection algorithm due to its network centrality, didn't let us deploy the SEG in this location. From our calculation, we could be able to reduce the data traffic around 871 MB compared to the results in Figure 17b. The main reason is that node would become a central hub for content distribution to the rest o the network: that would benefit everyone else except from that particular node. To summarize, in a mid-size network zone (e.g., 80 nodes) as in the Guifi-Sants case, if we can deploy the PiCasso in critical nodes of the network, the expected traffic reduction will be similar in percentage, but given the larger size of the network zone, having content still one hop away from users results in a bigger difference in terms of intra-network traffic that reduces network congestion, in addition to an increase of controller (server) load that grows more slowly than the size of the network.

6. Related Work

The work in PiCasso brings together many building blocks aiming at developing an efficient platform for service delivery in challenging network environments. This section reviews the main related works by classifying them in the following areas: edge computing platforms, content delivery networks (CDNs) and service placement. Each subsection discusses the works and compares with our work.

6.1. Edge Computing Platforms

Many researchers have leveraged the advantage of lightweight virtualisation technologies (e.g., Docker [17], Unikernel [26]) by proposing edge computing platforms to improve the QoS, security and privacy [27, 28, 29, 24, 8, 30, 31].

The work in [27] studies how Information-Centric Networking in combination with Mobile Edge Computing can work together in the context of connected vehicle environments. Authors present a vehicular scenario and list the challenges. However, the authors propose only the conceptual design architecture (without a proof of concept prototype). *Sathiaseelan et al.* [28] proposes Cloudrone, an edge computing platform for delivering the services over a cluster of flying drones. This work reports only a feasibility study and evaluation of the system using Docker containers over a single Raspberry Pi device. Similar to this work, *Yehia et al.* [29] study the scalability of Docker containers with different generations of Raspberry Pi's. Accordingly, these works are still lacking of required components for edge computing platform such as orchestration, monitoring and communication modules. The work in [32] presents a general framework where global cloud and ICN platforms are complemented by local clouds formed at the edge of the network by mobile devices. The prototype of PiCasso has been introduced in [33]. However, the evaluation of communication protocol for delivering the service has not been discussed yet. In contrast, this paper presents a complete architecture of PiCasso and evaluates the performance of service delivery with HANET algorithm and NDN solution. Cloudy [8] is the core software of the community clouds [2], as it unifies the different tools and services for the distributed cloud system with a Debian-based Linux distribution. Cloudy provides custom decentralised services for network management and service discovery. Paradrop [24] is a specific edge computing platform that provides (modest) computing and storage resources at the "extreme" edge of the network allowing third-party developers to flexibly create new types of services. The main limitation of these two platforms is a lack of service controller who automatically applies complex algorithms for service deployment regarding network condition and hardware resources. Furthermore, they are relied on host-centric communication which is not efficient for community networks as discussed in Section 4.2. Another work similar to ours is SCANDEX [30], a service centric networking framework for challenging decentralised networks by bringing together the lightweight virtulisation, ICN and DTN technologies. However, the authors propose only the conceptual design architecture. NFaaS [31] is another platform that aims to leverage the information-centric communication. NFaaS architecture is based on Unikernel and NDN while enabling the seamless execution of stateless microservices across the network. However, the authors only evaluate the system through simulation while the real implementation is still under development.RICE [34] is another edge-ICN platform that allows remote invocation to execute the function on the fly. RICE uses a TLV field of NDN message to add more information on the message which also supports the push communication without tweaking the natural pull based model of NDN.

6.2. Content Delivery Networks

With the massive growth of Internet traffic, CDNs has become a key solution to support scalability of Internet content delivery services. However, current CDNs are not collaborative, there is a lack of collaboration among CDNs providers (e.g., Akamai, Limelight) and ISPs. The leading content providers like Google, Amazon, Facebook also prefer to build their own CDNs infrastructure with private policies [35]. Despite there is an attempt from CDNi IETF working group proposing a collaborative solution for CDNs, the proposal just focuses on the protocol design rather than practical implementation [36]. Moreover, CDNs architecture is still based on host-centric communication whilst the CDN server can become easily congested and become a network bottleneck [37].

The clean slate approach called Information Centric Networks (ICNs) has recently emerged which inherently integrated the content delivery capability in the architecture [38]. Several research projects have been proposed to cope with the efficiency of content delivery, which have also been considered as the future Internet architecture [12, 13, 14, 15, 16]. Among those ICNs realisations, NDN (Named Data Networking) 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 [13] and RIFE [14] architectures are designed based on a centralised solution where there is a central entity to control the published and subscribed requests. In PiCasso, we have extended the NDN code base in order to leverage the distributed in-network caching in a network zone while integrating a new service abstraction layer to support service delivery rather than static content.

6.3. Service/Node Placement

Service placement is a key function of cloud management systems. By monitoring the resources on a system, service placement aims to balance load through the allocation, migration and replication of tasks. We look at the service placement problem in three different environments: wireless networks, data centres (DCs) and distributed data centres.

The work of Al Arnaut [39, 40], 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 first phase aims to partition the network into p sub graphs and each partition will have one replica node. The second phase aims to distribute the content to be cached in the replicas based on content popularity. Panadero et. al. [41, 42] proposes the Multi Criteria Biased Randomized (MCBR) method, a selection method for large-scale systems that use unreliable nodes. MCBR method is based on a multicriteria optimization strategy. They evaluate their method in a microblogging social network formed by a large number of microservices hosted by volunteer nodes. Selimi et. al., [43] put forward a service placement algorithm, called BASP, to place micro-cloud services in CMNs. The algorithm uses K-Means for clustering and a lightweight bandwidth computation/estimation heuristic. The HANET algorithm used in PiCasso platform is the advanced and lighthtweight version of the BASP. Coimbra et. al., [44] proposes a novel service placement approach based on community finding using a scalable graph label propagation technique and decentralized election procedure. Another example of a network-aware approach is the work from Moens in [45] which employs a Service Oriented Architecture (SOA), where applications are constructed as a collection of services. Their approach performs node and link mapping simultaneously. The work in [46] extends the work of Moens in wireless settings taking into account the IoT. Spinnewyn [47] provides a resilient placement of mission-critical applications on geo-distributed clouds using heuristic based on subgraph isomorphism detection. Tantawi [48, 49] uses biased statistical sampling methods for cloud workload placement. Regarding the service placement through migration, the authors in [50] and [51] study the dynamic service migration problem in mobile edge-clouds that host cloud-based services at the network edge. The work in [52] evaluates the migration performance of various real applications in mobile edge clouds (MEC). The work of Elmroth [53] takes into account rapid user mobility and resource cost when placing applications in Mobile Cloud Networks (MCN). Sevil et. al., [54] propose a fully automated approach to the joint optimization problem of scaling and placement of virtual network services.

Most of the work in the distributed data centers consider micro-datacenters, where in our case the CMN such as Guifi-Sants consist of constraint/low-power devices such as Raspberry Pi's. Our service placement heuristic HANET allows us to prioritize, in an easy and fast way the most important parameters of both the network and the nodes to place services, providing a flexible and agile method. Further, most of the above mentioned pieces of work are not applicable to our case because we have a strong heterogeneity given by the limited capacity of nodes and links, as well as asymmetric quality of wireless links.

7. Conclusion

The network environment in Community Mesh Networks (CMNs) is highly dynamic and network servers tend to be very modest, which make it hard to offer good quality content and services to the end-users. In this paper, we have analysed the characteristics of a Guifi.net network zone to identify the key requirements for developing edge computing and content services. From this analysis, we have argued that most of the existing platforms are not suitable for the CMNs since they rely on host-centric communication. In this aspect, we propose PiCasso, an edge computing platform that utilises the strength of lightweight virtualisation technology and Information-Centric Networking (ICN) to overcome the challenges in CMNs and deliver most content to users one or few hops away only. Unlike other platforms, PiCasso contains a decision engine that manages the service deployment operation in network zones. To address this issue, we have also proposed a heuristic algorithm, HANET, which considers both hardware and network resources. Through our results, HANET optimally selects the nodes to host the service and ensures that the end-users can achieve better QoS, depending mainly on the performance of a single hop to reach a SEG, instead of the multi-hop traffic to the origin server. Apart from improving QoS of end-users services, our results show that ICN plays a key role to improve the service delivery time as well as reducing the traffic consumption in CMNs.

There are several directions to extend this work. First, we plan to implement the user interface of service controller where the network administrators can manage the service deployment operation. This interface will include the option to select and update the service deployment algorithms for different scenarios. Second, we are developing several algorithms (e.g., based on centrality measures) that could support different scenarios and requirements for service deployment. Lastly, we wish to deploy PiCasso in other CMN zones which might have different environments.

Acknowledgments

This paper has been supported by the AmmbrTech Group, the Spanish government TIN2016-77836-C2-2-R and the European Community H2020 Programme netCommons (688768) and UMobile (645124). The authors would like to thank the people from the Guifi.net (Guifi-Sants) community network for hosting the servers and supporting the experiments. Thanks to Roger Pueyo for proofreading the article.

References

[1] D. Vega, R. Baig, L. Cerdà-Alabern, E. Medina, R. Meseguer, L. Navarro, A technological overview of the guifi.net community network, Computer Networks 93, Part 2 (2015) 260 – 278. doi:http://dx.doi.org/10.1016/j.comnet.2015.09.023.
URL //www.sciencedirect.com/science/article/pii/S1389128615003436

[2] M. Selimi, A. M. Khan, E. Dimogerontakis, F. Freitag, R. P. Centelles, Cloud services in the guifi.net community network, Computer Networks 93, Part 2 (2015) 373 – 388. doi:http://dx.doi.org/10.1016/j.comnet.2015.09.007.
URL //www.sciencedirect.com/science/article/pii/S1389128615003175

[3] R. Baig, R. P. Centelles, F. Freitag, L. Navarro, 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, 2017, pp. 31–36. doi:10.1109/GIIS.2017.8169801.
URL https://doi.org/10.1109/GIIS.2017.8169801

[4] A. Lertsinsrubtavee, M. Selimi, A. Sathiaseelan, L. Cerdà-Alabern, L. Navarro, J. Crowcroft, Information-centric multi-access edge computing platform for community mesh networks, in: Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies, COMPASS '18, ACM, New York, NY, USA, 2018, pp. 19:1–19:12. doi:10.1145/3209811.3209867.
URL http://doi.acm.org/10.1145/3209811.3209867

[5] A. Lertsinsrubtavee, A. Ali, C. Molina-Jimenez, A. Sathiaseelan, J. Crowcroft, Picasso: A lightweight edge computing platform, in: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), 2017, pp. 1–7. doi:10.1109/CloudNet.2017.8071529.

[6] A. Neumann, E. Lopez, L. Navarro, An evaluation of bmx6 for community wireless networks, in: 8th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 2012 I, 2012, pp. 651–658. doi:10.1109/WiMOB.2012.6379145.

[7] A. Neumann, E. Lopez, L. Cerda-Alabern, L. Navarro, Securely-entrusted multi-topology routing for community networks, in: 2016 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS), 2016, pp. 1–8.

[8] R. Baig, F. Freitag, L. Navarro, Cloudy in guifi.net: Establishing and sustaining a community cloud as open commons, Future Generation Computer Systemsdoi:https://doi.org/10.1016/j.future.2017.12.017.
URL http://www.sciencedirect.com/science/article/pii/S0167739X1732856X

[9] Guinux, https://guifi.net/en/node/29320, accessed: 2018-02-10.

[10] E. Dimogerontakis, R. Meseguer, L. Navarro, Internet Access for All: Assessing a Crowdsourced Web Proxy Service in a Community Network, Springer International Publishing, Cham, 2017, pp. 72–84. doi:10.1007/978-3-319-54328-4_6.
URL http://dx.doi.org/10.1007/978-3-319-54328-4_6

[11] D. Vega, L. Cerda-Alabern, L. Navarro, R. Meseguer, 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, 2012, pp. 612–619. doi:10.1109/WiMOB.2012.6379139.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, Networking named content, in: Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT '09, ACM, New York, NY, USA, 2009, pp. 1–12. doi:10.1145/1658939.1658941.
URL http://doi.acm.org/10.1145/1658939.1658941

[13] PURSUIT a Pub/Sub Internet, http://www.fp7-pursuit.eu/PursuitWeb/, accessed: 2018-02-10.

[14] RIFE: Architecture for an Internet for everybody, https://rife-project.eu/, accessed: 2018-02-10.

[15] Scalable and Adaptive Internet Solutions (SAIL), http://www.sail-project.eu, accessed: 2018-02-10.

[16] NetInf - Network of Information, http://www.netinf.org, accessed: 2018-02-10.

[17] Docker technology, https://www.docker.com/what-docker, accessed: 2018-02-10.

[18] C.-A. Sarros, A. Lertsinsrubtavee, C. Molina-Jimenez, K. Prasopoulos, S. Diamantopoulos, D. Vardalis, A. Sathiaseelan, 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, 2017, pp. 210–211. doi:10.1145/3125719.3132096.
URL http://doi.acm.org/10.1145/3125719.3132096

[19] U. De Silva, A. Lertsinsrubtavee, A. Sathiaseelan, C. Molina-Jimenez, K. Kanchanasut, Implementation and evaluation of an information centric-based smart lighting controller, in: Proceedings of the 12th Asian Internet Engineering Conference, AINTEC '16, 2016.

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

[21] "Apache Benchmarking tool".
URL `https://httpd.apache.org/docs/2.4/programs/ab.html`

[22] T. Palit, Y. Shen, M. Ferdman, Demystifying cloud benchmarking, in: 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2016, pp. 122–132.

[23] Introducing a powerful open source social networking engine, https://elgg.org/, accessed: 2018-02-10.

[24] P. Liu, D. Willis, S. Banerjee, Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge, in: 2016 IEEE/ACM Symposium on Edge Computing (SEC), Vol. 00, 2016, pp. 1–13. `doi:10.1109/SEC.2016.39`.
URL `doi.ieeecomputersociety.org/10.1109/SEC.2016.39`

[25] A. Afanasyev, NFD Developer's Guide, Tech. rep. (Feb. 2018).
URL `http://named-data.net/publications/techreports/`

[26] M. Anil, D. J. Scott, Unikernels: Rise of the Virtual Library Operating System, Queue 11 (11) (2013) 30:30–30:44.

[27] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, D. Kutscher, Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions, in: Proceedings of the Workshop on Mobile Edge Communications, MECOMM '17, ACM, New York, NY, USA, 2017, pp. 7–12. `doi:10.1145/3098208.3098210`.
URL `http://doi.acm.org/10.1145/3098208.3098210`

[28] A. Sathiaseelan, A. Lertsinsrubtavee, A. Jagan, P. Baskaran, J. Crowcroft, Cloudrone: Micro clouds in the sky, in: Proc. 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (DroNet'16), 2016.

[29] Y. Elkhatib, B. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, E. Rivière, On using micro-clouds to deliver the fog, IEEE Internet Computing 21 (2) (2017) 8–15. `doi:10.1109/MIC.2017.35`.

[30] A. Sathiaseelan, L. Wang, A. Aucinas, G. Tyson, J. Crowcroft, Scandex: Service centric networking for challenged decentralised networks, in: Proc. 2015 Workshop on Do-it-yourself Networking: an Interdisciplinary Approach (DIYNetworking '15), 2015.

[31] M. Król, I. Psaras, Nfaas: Named function as a service, in: Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN '17, ACM, New York, NY, USA, 2017, pp. 134–144. `doi:10.1145/3125719.3125727`.
URL `http://doi.acm.org/10.1145/3125719.3125727`

[32] E. Borgia, R. Bruno, M. Conti, D. Mascitti, A. Passarella, Mobile edge clouds for information-centric iot services, in: 2016 IEEE Symposium on Computers and Communication (ISCC), 2016, pp. 422–428. `doi:10.1109/ISCC.2016.7543776`.

[33] A. Lertsinsrubtavee, A. Ali, C. Molina-Jimenez, A. Sathiaseelan, J. Crowcroft, Picasso: A lightweight edge computing platform, in: Proceedings of the 6th IEEE International Conference on Cloud Networking, CloudNet'17, 2017.

[34] M. Król, K. Habak, D. Oran, D. Kutscher, I. Psaras, Rice: Remote method invocation in icn, in: Proceedings of the 5th ACM Conference on Information-Centric Networking, ICN '18, ACM, New York, NY, USA, 2018, pp. 1–11. `doi:10.1145/3267955.3267956`.
URL `http://doi.acm.org/10.1145/3267955.3267956`

[35] G. Carofiglio, G. Morabito, L. Muscariello, I. Solis, M. Varvello, From content delivery today to information centric networking, Comput. Netw. 57 (16) (2013) 3116–3127.

[36] L. Peterson, B. Davie, R. van Brandenburg, Framework for Content Distribution Network Interconnection (CDNI), RFC 7336 (Aug. 2014).
URL `https://tools.ietf.org/html/rfc7336`

[37] Q. Jia, R. Xie, T. Huang, J. Liu, Y. Liu, The collaboration for content delivery and network infrastructures: A survey, IEEE Access 5 (2015) 18088 – 18106. `doi:10.1109/ACCESS.2017.2715824`.

[38] G. Xylomenos, C. N. Ververidis, V. A. S. andn Nikos Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, G. C. Polyzos, A survey of information-centric networking research, IEEE Communications Surveys & Tutorials 16 (2) (2014) 1024–1049.

[39] Z. Al-Arnaout, Q. Fu, M. Frean, 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, 2012, pp. 39–44. `doi:10.1145/2229087.2229098`.
URL `http://doi.acm.org/10.1145/2229087.2229098`

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

[41] J. Panadero, J. de Armas, X. Serra, J. M. Marquès, Multi criteria biased randomized method for resource allocation in distributed systems: Application in a volunteer computing system, Future Generation Computer Systems 82 (2018) 29 – 40. `doi:https://doi.org/10.1016/j.future.2017.11.039`.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X17315236`

[42] J. Panadero, L. Calvet, J. M. Marquès, A. A. Juan, A simheuristic approach for resource allocation in volunteer computing, in: 2017 Winter Simulation Conference (WSC), 2017, pp. 1479–1490. `doi:10.1109/WSC.2017.8247890`.

[43] M. Selimi, L. Cerdà-Alabern, F. Freitag, L. Veiga, A. Sathiaseelan, J. Crowcroft, A lightweight service placement approach for community network micro-clouds, Journal of Grid Computing`doi:10.1007/s10723-018-9437-3`.
URL `https://doi.org/10.1007/s10723-018-9437-3`

[44] M. E. Coimbra, M. Selimi, A. P. Francisco, F. Freitag, L. Veiga, Gelly-scheduling: Distributed graph processing for service placement in community networks, in: 33rd ACM/SIGAPP Symposium On Applied Computing (SAC 2018), ACM, 2018.

[45] H. Moens, et al., Hierarchical network-aware placement of service oriented applications in clouds, in: 2014 IEEE Network Operations and Management Symposium (NOMS), 2014, pp. 1–8. `doi:10.1109/NOMS.2014.6838230`.

[46] B. Spinnewyn, B. Braem, S. Latré, Fault-tolerant application placement in heterogeneous cloud environments, in: Network and Service Management (CNSM), 2015, pp. 192–200. `doi:10.1109/CNSM.2015.7367359`.

[47] B. Spinnewyn, R. Mennes, J. F. Botero, S. Latré, Resilient application placement for geo-distributed cloud networks, Journal of Network and Computer Applications 85 (2017) 14 – 31, intelligent Systems for Heterogeneous Networks. doi:https://doi.org/10.1016/j.jnca.2016.12.015.
URL http://www.sciencedirect.com/science/article/pii/S1084804516303149

[48] A. N. Tantawi, Solution biasing for optimized cloud workload placement, in: 2016 IEEE International Conference on Autonomic Computing (ICAC), 2016, pp. 105–110. doi:10.1109/ICAC.2016.34.

[49] A. N. Tantawi, 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, 2015, pp. 195–210. doi:10.1007/978-3-319-22264-6_13.
URL http://dx.doi.org/10.1007/978-3-319-22264-6_13

[50] R. Urgaonkar, S. Wang, T. He, M. Zafer, K. Chan, K. K. Leung, Dynamic service migration and workload scheduling in edge-clouds, Performance Evaluation 91 (2015) 205 – 228. doi:http://dx.doi.org/10.1016/j.peva.2015.06.013.
URL //www.sciencedirect.com/science/article/pii/S0166531615000619

[51] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, K. K. Leung, Dynamic service placement for mobile micro-clouds with predicted future costs, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2017) 1002–1016. doi:10.1109/TPDS.2016.2604814.
URL https://doi.org/10.1109/TPDS.2016.2604814

[52] A. Machen, S. Wang, K. K. Leung, B. J. Ko, T. Salonidis, Live service migration in mobile edge clouds, in: IEEE Wireless Communications, 2017.

[53] W. Tärneberg, A. Mehta, E. Wadbro, J. Tordsson, J. Eker, M. Kihl, E. Elmroth, Dynamic application placement in the mobile cloud network, Future Generation Computer Systems 70 (2017) 163 – 177. doi:http://dx.doi.org/10.1016/j.future.2016.06.021.
URL //www.sciencedirect.com/science/article/pii/S0167739X16302060

[54] S. Draxler, H. Karl, Z. A. Mann, Joint optimization of scaling and placement of virtual network services, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), 2017, pp. 365–370. doi:10.1109/CCGRID.2017.25.