DIMON: Distributed Monitoring System for Decentralized Edge Clouds in Guifi.net

Roger Pueyo Centelles∗, Mennan Selimi†, Felix Freitag∗, Leandro Navarro∗
∗Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
†Max van der Stoel Institute, South East European University, North Macedonia

Abstract—Community-built telecommunication networks such as Guifi.net demonstrate how end users can actively collaborate in the self-provision of network services, for instance by operating a self-organized distributed monitoring system. Network monitoring is performed by many small servers at the users’ premises but data are only accessible via a centralized interface. Besides, due to network partitions and churn of the monitoring servers, failures in the monitoring system are frequent, leaving parts of the network unmonitored. Distributed databases are a promising solution for data replication under network partition condition, but they suffer from a trade-off between data consistency and availability. Furthermore, these databases are used in data centers with abundant computing resources, not in light edge networks.

In this work we present DIMON, a reliable edge-based, eventually-consistent monitoring system that leverages CRDT-based data structures implemented in AntidoteDB. Conflict-free replicated data types (CRDTs) are able to converge to a consistent state in environments with network partitions as those found in edge networks. Our results give insights on the load of AntidoteDB on edge devices under different scenarios of read and write operations. The experiments carried out in a production network with a real system implemented contribute to the research community’s knowledge about the available technologies for a consistent replicated data storage layer to support edge computing clouds.

Index Terms—distributed monitoring; edge clouds; CRDT;

I. INTRODUCTION

Community networks such as Guifi.net1 are bottom-up, citizenship-driven technological, social and economic projects with the objective of creating a free, open and neutral telecommunications network based on a commons model [1]. The whole infrastructure can be understood as a crowd-sourced, multi-tenant collection of heterogeneous network devices – wired and wireless– with IP addressing, interconnected between them and forming a partially-meshed network. Figure 1 captures Guifi.net around Barcelona and surrounding towns, although the network expands over several countries. Guifi.net is considered to be the world’s largest wireless community network with more than 35,000 operational nodes.

Edge computing builds upon the advantages of cloud computing, but extends the traditional cloud services in data centers with the capacities of local processing [2]. For this, lightweight computing devices (e.g., IoT gateways) are placed at the edge of the networks, where the data are produced. Local processing, on edge devices, improves response times of cloud services and requires less data to be transmitted to the cloud, reducing bandwidth consumption. Edge computing solutions are already operational in many industrial and consumer-oriented scenarios, provided by major Internet service providers and customized by specialized enterprises.

The edge computing model in Guifi.net is different to the approach above. Owners of edge devices are enabled to collaborate and actively participate in the provision of services, and contribute to sustain edge micro-clouds [3], [4]. Their goal is to form a scattered cloud with user-contributed computing and communication resources that allow providing services of local interest over few network hops. Services may include providing Internet access through proxies [5], but also applications deployed within the community network [6]. This concept perfectly matches the geographical distribution (Figure 1) and multi-tenancy of the underlying Guifi.net infrastructure.

The cloud infrastructure for edge computing in Guifi.net is located at the network edge. Most devices that contribute to the resource pool are located at the users’ homes or in the premises of municipalities. The software platform installed on these devices is the Cloudy platform [7], which is open and can be extended with additional services by the participants. The hardware used to form the infrastructure is heterogeneous, ranging from Single-Board-Computers (SBCs), such as the Raspberry Pi, to minicomputers (Figure 2) and desktop PCs. This approach for edge computing in Guifi.net started to be researched and developed in the last few years and nowadays, in 2019, accounts for several dozens of operational devices in the community network.

The current monitoring system in Guifi.net is built around a centralized database that lists all the devices in the network. A number of servers, which are geographically spread all over the network, take care of monitoring the network nodes. Each

1 What is Guifi.net? - https://guifi.net/en/what_is_guifinet
monitoring server periodically fetches the list of its assigned devices from the central database; then it checks for their status and gathers information about their network interfaces, etc. The assignment of a network device to its monitoring server is mostly static and rarely changes over time [3], [8]. All the information collected stays local to the different monitoring servers, not being automatically replicated or distributed between them or elsewhere. This current system fulfills most of the basic monitoring needs, but it has several shortcomings, as it does not leverage technologies for automation, distribution of the workload and decentralization of coordination and decision-making about the allocation of the monitoring tasks. To name a few of the limitations of the current system:

- No redundancy: every device is monitored by a single server.
- No load balancing between monitoring servers.
- No automatic detection of server failures and reassignment of devices to another monitoring server.
- No replication or distribution of collected data.
- Need for manual intervention: devices usually require to be manually assigned to a specific monitoring server.

The contribution of this paper is the design and deployment experience of an edge-based eventually consistent and reliable monitoring system codenamed DIMON, supporting distributed and consistent concurrent and replicated read and writes. DIMON monitoring platform’s approach leverages CRDT-based distributed data structures [9] implemented in AntidoteDB [10]. CRDTs are applied in our scenario since they are able to converge to a consistent state in environments with network partitions as those found in edge networks. It is also worth mentioning that the design supports the distributed coordination of monitoring servers in performing the assignment of network devices across them in a decentralized way. Furthermore, using AntidoteDB’s features allows to automate the replication and distribution across the different servers in a decentralized way, while keeping data persistent, consistent and reliably stored. Generalizing our results, the evaluated technology can offer a shared data layer for the network edge, upon which advanced edge computing applications going beyond monitoring systems can be built upon.

The rest of the paper is organized as follows. In Section II we explain the current [legacy] monitoring system used in Guifi.net. Section III describes DIMON, our proposed new monitoring system. Section IV presents a functional evaluation of the monitoring system in a production network of Guifi.net. Section V describes related work and Section VI concludes and discusses future research directions.

II. LEGACY MONITORING SYSTEM

The monitoring system nowadays in production in Guifi.net is built around a central MySQL database (DB), coupled with the Guifi.net website, which lists all the nodes in the network (i.e., network devices, such as routers) and assigns them to the servers spread all over the network in charge of monitoring them. The current system has three main shortcomings that make it fragile: (i) each router is monitored by only one server and (ii) when a monitoring server goes down, this is not automatically reported to the Guifi.net website or the central DB, so network nodes are left unmonitored because they are not automatically reassigned to another server, and (iii) data collected about one node are stored only by a single monitoring server.

The current Guifi.net monitoring tool, called SNP Services, was developed by the community, around Guifi.net’s central DB and website, where the actual assignment of network nodes to the different monitoring servers occurs. Each SNP Services instance (i.e., each monitoring server) periodically fetches an updated list with the nodes it has been assigned. The server will only monitor the nodes in the list, ignoring the rest of nodes in the network, no matter how close of far they are, its current workload, etc. The information collected stays local to the servers, not being automatically replicated or distributed anywhere else.

Currently there are about 200 monitoring servers known or considered to be active. These devices are low- to mid-end power devices, ranging from embedded ARM computers like the Raspberry Pi, low-power x86 Intel Atom-based devices like the Minix NEO-Z64, x86 virtual machines to refurbished bare-metal machines. The servers are spread all over the Guifi.net network in an organic way, usually without a carefully-planned distribution. Monitoring servers are connected to the Guifi.net network through their local node; the physical connection is performed via cable (Ethernet). Guifi.net nodes are interconnected by a mix of mid- to long-distance wireless links (WiFi) and wired (fiber optics) links.

III. THE DIMON MONITORING SYSTEM

The new monitoring system DIMON aims at solving the limitations of the legacy monitoring system and provide comprehensive and reliable monitoring data for all network devices despite network and server failures. To achieve that, we describe the use-cases, requirements, resulting architecture, implementation algorithm and data model.

A. Use-Cases and Requirements

Figure 3 provides a high-level overview of use cases in the new monitoring system. A number (n) of monitoring servers, drawn on the top left corner, are the primary actors that perform actions within the system. These servers are composed by two items, the assignment and the monitoring components.
The new monitoring service has functional requirements regarding what a monitoring system should provide, which consist in a set of read and write operations to be carried out by the different primary and secondary actors. The required services rely on a distributed storage service.

The assignment component within each monitoring server carries out the mapping of the networking devices assigned to itself. For this, the assignment component shall read the current data of the assignments of all monitoring servers from a distributed storage service in order to obtain the latest assignments. In order to take precise decisions, the assignment component shall obtain consistent data from the distributed storage service. Further, the assignment component shall write new assignment data, updated with its own assignment—decided by itself—to the distributed storage service in order to communicate to the other monitoring server its latest assignment decisions taken. The decision that an assignment component takes on its updated assignment can be extended by taking into account contextual information and additional decision support functions.

The monitoring component generates the monitoring data from observation of the networking devices (e.g., routers) and store them in the distributed storage service. If several data sets are taken for the same networking device, simultaneously, by different servers, the stored data shall be aggregated or merged.

Other actors are the Guifi.net database (GuifiDB), which supports in the initialization of the assignments. For this, the GuifiDB shall provide information about the available monitoring servers and network devices to be written to the distributed storage service. Information about the network state and about network devices can feed contextual information. A distributed database shall provide the storage service to support the different read and write actions.

The new monitoring system has the following non-functional requirements:

- Automated assignment: no need for manual intervention to assign monitoring servers to network devices. The system shall do the assignment automatically. In a permanent operation these services should run autonomously without manual intervention.
- Automated reconfiguration: automatic detection of server failures and reassignment of network devices to another monitoring server. The system shall carry out the reconfiguration automatically. Each monitoring server responds to the situation of the assignment of monitoring servers to network devices and makes a decision to compensate failures of servers or network devices.
- Redundancy: every networking device is monitored by several servers (i.e., the monitoring servers check which network devices have less monitors and decide autonomously to become a monitor for any of these devices).
- Load balancing between servers: assignment decisions take into account server load. The load of each monitoring service should not significantly affect the user experience if the device is used for multiple services.
- Data replication: the collected data is replicated or distributed. In the event of network partition or churn of some storage nodes the data should still be available for being retrieved by the monitoring service.

The results can bring valuable feedback to the research on the distributed storage service design. The specific performance metrics and evaluation targets/criteria we expect to apply in the experimental evaluation of the network monitoring use case implementation are further explained in Section IV.

B. Architecture and Implementation

The requirements of the monitoring use case include concurrent reads and writes into a storage system from geographically distributed monitoring servers. We have singled out AntidoteDB\(^2\) as a distributed CRDT-based data store which principally is able to support or fulfill these functional requirements.

The general architecture of DIMON is depicted in Figure 4, showing the monitoring servers and their integration with the Guifi.net website and network nodes. On the top of the picture appears the Guifi.net website and its central DB, which contains the lists of network nodes and monitoring servers. These two lists are always provided by the website, and can be considered to be correct and readily available at any time. Compared to the current implementation (the legacy monitoring system), the main difference lays on the proposed monitoring servers ⇔ network devices mapping, which is not generated centrally, but in a distributed and collaborative way, by the monitoring servers themselves. To achieve this, all the monitoring servers run a local instance of AntidoteDB.

The monitoring servers spread around the network share a distributed data space, implemented by means of AntidoteDB. The servers use this data space to know which is the current monitoring servers ⇔ network devices mapping and update it according to different criteria, making sure that all nodes are being actively monitored and reacting to eventual failures (de-commissioned servers, network partitions, etc.). Additionally, the main Guifi.net web and DB server also runs an Antidote instance, connected and synchronized with those running in the monitoring servers. This instance is used as the single authoritative entry point for updates on the nodes list. The list may also contain additional details or information about

\(^2\) AntidoteDB: https://www.antidotedb.eu/
network nodes (i.e. their role in terms of network topology, performance or economic interest) in order to prioritize its monitoring or dedicate more resources to it.

The monitoring servers, once they have the list of nodes to watch, coordinate between them in order to perform the actual monitoring of all nodes. First, they assign every single node to –at least– one monitoring server (Phase 2 of Algorithm 1). This task can be performed in many ways (for instance, each monitoring server could start picking, at random, nodes not yet assigned and assign them to itself). Another method could be that monitoring servers probe their reachability and RTT to every network node to then link each node to its closest monitoring server (or the closer ones). In the context of a large community network like Guifi.net, it is possible to observe frequent changes in the network topology as the dynamic routing protocols react to the status of the links between the nodes (link quality, RTT, throughput, usage, etc.). Despite unlikely, network partitions may also eventually appear, even if for short periods of time.

C. Data model

The data manipulated by the DIMON monitoring system consists of two sets of objects and a mapping between these objects. A simplified depiction of the data model and the application components involved in data manipulation is shown in Figure 4. Further, the detailed algorithm of the monitor-assign function is depicted in Algorithm 1.

The first set contains a list with all the nodes in Guifi.net that have to be monitored. All the Guifi.net nodes are identified by a unique numeric ID (e.g. 58266), which remains immutable through all its lifespan. Additional information, such as an associated IPv4 address (e.g. 10.1.33.33) may be attached as a string-structured JSON item. The current nodes list contains around 35,000 nodes, and grows at a rate of 25 nodes per day. Objects in this list are immutable (each network node is identified by its unique ID, which does not change through all its lifespan). The data in this first set is only modified by authoritative updates issued from the Guifi.net website; the monitoring servers only read it but do not modify it.

The second set contains a list with all the active monitoring servers. Servers are also identified by a unique numeric ID, being the servers list a subset of the nodes list (a monitoring server is, indeed, a device in the network that must be monitored too). The data in this second set is only modified by authoritative updates issued from the Guifi.net website; the monitoring servers only read it but do not modify it.

The mapping between the nodes list and the servers list can be seen as a collection of relations between objects, one in each set (one network node and one monitoring server). Any monitoring server may modify the mapping between nodes and servers (add, update or remove these relations at any time). According to different criteria –such as current workload, network status and other– each monitoring server will, for instance, assign itself a number of nodes and will update the monitoring servers ⇔ network devices mapping accordingly. This assignment may change over time, as new nodes are added to the list, network conditions change, workload is redistributed, monitoring servers join or exit the pool, etc.

Given the nature of the application, and in order to successfully deal with concurrent updates of the mapping, eventual data consistency and integrity between the different DB instances are required. By leveraging these properties, it can be ensured that all network nodes end up being properly assigned to monitoring servers.

IV. EVALUATION

The aim of this section is to provide a functional and feasibility evaluation of the DIMON monitoring system explained...
Algorithm 1 Monitor-Assign Algorithm

Require:
- dbhost  ➔ Hostname/IP address of the AntidoteDB
- dbPort  ➔ TCP port of the AntidoteDB
- id      ➔ The unique ID of the monitor in the network
- minMonitors ➔ Min # of monitors a device needs
- maxDevices ➔ Max # of devices the monitor can watch

Phase 1 – Monitor registration

1: procedure REGISTRATION
2:     monitorsList[] ← GetGlobalMonitorsList
3:     AddMonitorToList(id, monitorsList[])
4:     UpdateGlobalMonitorList(monitorList[])
5: end procedure

Phase 2 – Monitor Self-assignment

6: procedure ASSIGNED(id)
7:     numDevices ← 0
8:     devicesInAntidote[] ← GetDevicesInAntidote()
9:     for each deviceInAntidote in devicesInAntidote[] do
10:        if (id is in deviceInAntidote.monitors[]) then
11:           numDevices++
12:        end if
13:     end for
14:     for each deviceInAntidote in devicesInAntidote[] do
15:        if (sizeOf(deviceInAntidote.monitors[]) < numDevices) && (numDevices < maxDevices) then
16:           assignMonitorToDevice(id, deviceInAntidote)
17:        end if
18:     end for
19: end procedure

Phase 3 – Monitor Self-unassignment

20: procedure UNASSIGNED(id)
21:     devicesInAntidote[] ← GetDevicesInAntidote()
22:     for each deviceInAntidote in devicesInAntidote[] do
23:        if deviceInAntidote.monitors[] > minMonitors then
24:           unassignMonitorFromDevice(id, deviceInAntidote)
25:        end if
26:     end for
27: end procedure

Phase 4 – Global assignment sanitization

28: procedure SANITIZE
29:     monitorsList[] ← GetGlobalMonitorsList
30:     localList ← GetGlobalMonitorList
31:     for each deviceInAntidote in devicesInAntidote[] do
32:        if each monitor in deviceInAntidote.monitors[] do
33:           if (monitor is not in monitorsList[]) then
34:              unassignMonitorFromDevice(monitor, deviceInAntidote)
35:           end if
36:        end for
37:     end for
38: end procedure

Phase 5 – Monitor Self-deregistration

39: procedure DeregISTRATION
40:     monitorsList[] ← GetGlobalMonitorsList
41:     RemoveMonitorFromList(id, monitorsList[])
42:     UpdateGlobalMonitorList(monitorList[])
43: end procedure

in Section III. To do this, we deploy the monitoring system on mini-PCs and server nodes in a production wireless mesh network called qMpsants³, which is part of Guifi.net. The expected outcome is to obtain AntidoteDB usage experience in edge environments using distributed monitoring as a use-case, to be shared with the users and developers community.

³ http://dsg.ac.upc.edu/qmpsu/index.php

Therefore, we focus on the resource usage including traffic overhead, CPU and memory usage, number of AntidoteDB operations etc.

A. Testbed

Our testbed aims to support the experimental evaluation of the monitoring system in the Guifi.net environment. For this purpose, we have installed and configured a set of devices and machines connected to the Guifi.net network on which we deploy the DIMON system. Monitoring servers in Guifi.net are often installed as a service on a diversity of hardware, which can range from resource-constraint SBCs to desktop servers. Devices are often purpose not only for monitoring, but they run several other services, be them related to the network or to user applications.

For the initial evaluation, we have chosen to build a testbed with a set of resource-constraint physical nodes and virtual machines. The physical nodes are edge devices which host AntidoteDB and consist of four MINIX Z83-4 mini-PCs (Minix Cluster) connected to a 100 Mbps switch as shown in Figure 5. The Minix devices are running Debian Stretch and have an Intel Atom x5-Z8350 4-cores CPU @ 1.44 GHz, 4 GB of DDR3L RAM and 32 GB eMMC. Further, for comparison purposes we have set up also a Proxmox VM Cluster: 6 VMs hosted in 6 clustered identical machines, connected to a 100 Mbps switch. Each VM is running Debian Stretch and is provided with 4 Intel i7-3770 CPU cores @ 3.4 GHz, 2 GB of DDR3 RAM and 30 GB of HD. We run one AntidoteDB instance (inside a Docker container) per Minix device and VM.

B. Main Features

In our experimental evaluation, we are focusing on two of the main phases (i.e., processes) of the DIMON system:

- **Monitor-Fetch**: This process parses a specified CNML (Community Network Markup Language) file and pushes its contents to AntidoteDB. The whole Guifi.net network description is included in a public cnml.xml file, which can be downloaded from the Guifi.net website, and is used by default. For our experimental use-cases we are using three different CNML files: the UPC sub-network (58 devices), the Barcelona sub-network (1602 devices) and the whole Guifi.net (39998 devices). There is a single fetch instance in the system, with authoritative writes/updates.
Monitor-assign is more operation intensive since it involves both monitor-fetch and monitor-assign. The complexity is shown in Figure 6, which reveals that the maximum synchronisation cost reached is 2100 kbps. Starting 2 (Minix 4) starts executing the monitor-assign function. The synchronisation cost reached is 2000 kbps. In Phase 3, Client 3 starts executing the monitor-assign function. The maximum synchronisation cost reached is 180 kbps. In Phase 1, Client 1 (Minix 1) starts executing the monitor-assign function. The average synchronisation cost (i.e., average incoming and outgoing traffic) between AntidoteDB instances (i.e., docker containers) is around 180 kbps. In Phase 1, Client 1 (Minix 3) starts executing the monitor-assign function. The maximum synchronisation cost reached is 2000 kbps. In Phase 3, Client 2 (Minix 4) start executing the monitor-assign function. The maximum synchronisation cost reached is 2100 kbps. Starting from Phase 4, both of the clients are writing concurrently to the Minix 1 and 2, and the maximum aggregated traffic (sync cost) generated is 4500 kbps.

**Resource Consumption:** In order to understand the resource usage of the DIMON monitoring platform, we measure CPU load and memory usage of monitoring servers when executing the two main processes of the system: monitor-fetch and monitor-assign.

Figures 9 and 10 depict the CPU load and memory usage when executing the fetch process of the DIMON monitoring system in one of the devices. During this operation, first we fetch three different database files (with different number of nodes) from the Guifi.net page, we parse them and push their contents to the local instance of the replicated AntidoteDB cluster.

When performing the fetch operation, the maximum CPU load reached is 26% in the Proxmox VMs and 42% in the Minix devices. Since the VMs have better CPUs they incur in less processing load to do the fetch operation, resulting in a lower CPU utilization. Minix devices show a significant smaller memory usage (max 140 MB) than VMs (max 200 MB), probably due to faster eMMC-based file access. Since the fetch operation is not a frequent one (i.e., it is run on a daily/weekly basis to incorporate new network devices), the CPU and memory values seem to be acceptable for commodity hardware.

Figure 11 shows the CPU load and memory usage of the assign process in the Minix cluster during 5 hours of continuous execution. Figure reveals that the assign process is more CPU and memory intensive than the fetch one. During this time, each monitoring server assigns itself a number of nodes to monitor (e.g., 100) and updates the monitoring servers network devices mapping accordingly. These periodic updates to AntidoteDB can be seen in the memory usage as spikes. The maximum CPU utilization reached is 180% i.e., almost two CPU cores because of frequent read and write operations.

### C. Results

**Number of Operations:** Figure 6 gives an overview in terms of operation complexity of two considered features (monitor-fetch and monitor-assign). The complexity is shown by number of operations necessary to execute functions. In the time-frame of 20 min, Figure 6 reveals that the number of operations during monitor-assign outnumbers the number of operations during monitor-fetch (i.e, 400 operations vs. 170). Monitor-assigns is more operation intensive since it is assigning, de-assigning devices to monitors constantly. Further, Figure 7 depicts number of open transactions when performing monitor-assign and monitor-fetch operation.

**Synchronisation Cost:** Figure 8 depicts the synchronisation cost i.e., the traffic overhead caused when running the monitor-assign function. Figure shows four distributed datacenters where two of them (Minix 3 and Minix 4) are acting as clients writing to Minix 1 and Minix 2, respectively. For easier explanation, we divide the time-frame into six phases as shown in Figure 7. In Phase 1, no monitoring system is running and the average synchronisation cost (i.e., average incoming and outgoing traffic) between AntidoteDB instances (i.e., docker containers) is around 180 kbps. In Phase 1, Client 1 (Minix 3) starts executing the monitor-assign function. The maximum synchronisation cost reached is 2000 kbps. In Phase 3, Client 2 (Minix 4) start executing the monitor-assign function. The maximum synchronisation cost reached is 2100 kbps. Starting from Phase 4, both of the clients are writing concurrently to the Minix 1 and 2, and the maximum aggregated traffic (sync cost) generated is 4500 kbps.

**Monitoring:** There are many commercial solutions running on top of public and private clouds that monitor virtual cloud usage (e.g., resource consumption and network performance). For instance, Amazon CloudWatch [11] is a monitoring and management service that monitors virtual resources of users such as Amazon EC2 instances. IBM Tivoli Monitoring [12] and HP Open View [13] are other monitoring systems aiming to optimize the performance and availability of IT infrastructures by focusing on the physical resources. Graphite [14] is an enterprise-ready monitoring tool that runs equally well on cheap hardware or Cloud infrastructure. Teams use graphite to track the performance of their websites, applications, business services, and networked servers. GMonE [15] is a general-purpose cloud monitoring tool which proposes a unified cloud monitoring taxonomy based on which it defines a layered cloud monitoring architecture. PCMONS [16] is a private cloud monitoring system that can be adapted for

---

4<https://www.cpubenchmark.net/compare/Intel-i7-3770-vs-Intel-Atom-x5-Z8350/096v32774>
Remote Client Minix 3 started
Idle phase
Monitoring system not started
Remote Clients 3 and 4
Concurrently Assign
Synchronization-cost
when two remote clients
execute monitor-assign
Remote Client Minix 4 started
Idle phase
Monitoring system not started
Remote Client 3 and 4
Concurrently Assign
Synchronization-cost
when two remote clients
execute monitor-assign

Figure 8. Synchronization cost (traffic overhead). Two remote clients (Minix 3 and Minix 4) writing concurrently to Minix 1 and Minix 2.

Figure 9. *fetch*: CPU and memory usage (Proxmox VM Cluster).

Figure 10. *fetch*: CPU and memory usage (Minix Cluster).

PCMONS does not collect the information needed to map virtual resources to physical resources. MonPaaS [17] is an open source adaptive monitoring platform as a service (MonPaaS). MonPaas integrates Nagios [18] and OpenStack. MonPaas monitors physical and virtual resources and also updates any change in physical or virtual infrastructure. The disadvantage of MonPaas is that it consumes extra physical resources.

**CRDT-based Distributed Databases:** Many works have integrated CRDTs in several storage systems that make them available to applications. The applications uses these CRDTs to
store their data, being the responsibility of the storage systems to synchronize the multiple replicas. The following commercial and academic prototypes systems use CRDTs: Azure Cosmos DB [19], Riak [20], Akka [21] Antidote [22] etc.

In contrast to these state-of-the-art approaches mostly focusing on specific issues in cloud monitoring, our main goal is to present a lightweight monitoring platform (with low overhead) that leverages CRDT based data structures to be used by a wide variety of applications in decentralized edge clouds. Furthermore, most of the related works mentioned above consider datacenters or micro-data centers as their environment, where in our case a community network such as Guifi.net consist of constrained/low-power devices such as Raspberry Pi boards, home gateways, mini PCs, etc.

VI. CONCLUSION AND OUTLOOK

Edge cloud computing will move from the current vertical architecture, where an edge node interoperates with a large cloud data center, to a horizontal architecture, where edge nodes will form microclouds. For this scenario, a distributed consistent data storage at the edge to enable the coordination of edge nodes for diverse applications seems to be promising.

In order to understand the performance of the technology option for such a storage service hosted on edge devices, this papers developed and evaluated the use case of an edge-based distributed monitoring system. The requirements of the monitoring system included the ability to perform concurrent and consistent writes to a shared distributed database from distributed clients. As a solution, CRDT data types implemented by the AntidoteDB database were used in order to leverage its synchronization capabilities for reaching consistent data.

The following results were obtained from an experimental deployment on real edge nodes: First, concurrent writes were successfully carried out to the database from multiple locations at the edge with a number of workloads of different intensity. Secondly, we observed that the CPU and memory usage of the monitoring system on the edge nodes occupied around one forth of its resources for carrying out permanent operations of specific workloads. The obtained results gave new insights on the technology options to build distributed edge-based storage systems.

The generalization of the design of our monitoring system can be interpreted as a coordination mechanism for decentralized nodes over a shared and eventually consistent data storage layer. We anticipate the applicability of this approach in edge computing applications beyond the presented use case. In the proliferation of edge nodes in the future, common ownership of these nodes will not be typical anymore, but rather there will be multiple nodes with different administrators that need to cooperate with each other. A data storage layer that runs at the edge as in the presented use case may be the basis to enable this interoperability and therefore gives new opportunities for edge and pervasive computing.

Our future work will focus on mechanisms for the intelligent assignment of monitoring servers, based on network conditions, workloads and predictions. We expect to see a trade-off between facilitating smart decisions that benefit the optimal resource allocation and the computing overhead required to enable, by the availability of real time and eventually consistent data, these smart decisions.

ACKNOWLEDGMENT

This work was supported by the European H2020 framework programme project LightKone (H2020-732505), by the Spanish government under contract TIN2016-77836-C2-2-R and the Catalan government AGAUR SGR 990.

REFERENCES