A Monitoring System for Distributed Edge Infrastructures with Decentralized Coordination

Roger Pueyo Centelles[1], Mennan Selimi[1,2], Felix Freitag[1(✉)], and Leandro Navarro[1]

[1] Universitat Politècnica de Catalunya, BarcelonaTech, Barcelona, Spain
{rpueyo,mselimi,felix,leandro}@ac.upc.edu

[2] Max van der Stoel Institute, South East European University, Tetovo, North Macedonia
m.selimi@seeu.edu.mk

Abstract. We present the case of monitoring a decentralized and crowd-sourced network infrastructure, that needs to be monitored over geographically distributed devices at the network edge. It is a characteristic of the target environment that both, the infrastructure to be monitored and the hosts where the monitoring system runs, change over time, and network partitions may happen. The proposed monitoring system is decentralized, and monitoring servers coordinate their actions through an eventually consistent data storage layer deployed at the network edge. We developed a proof-of-concept implementation, which leverages CRDT-based data types provided by AntidoteDB. Our evaluation focuses on the understanding of the continuously updated mapping of monitoring server to network devices, specifically on the effects of different policies for each individual monitoring server to decide on which and how many network devices to monitor. One of the policies is experimented by means of a deployment on 8 real nodes, leveraging the data replication of AntidoteDB in a realistic setting. The observed effects of the different policies are interpreted from the point of view of the trade-off between resource consumption and redundancy.

Keywords: Edge computing · Distributed monitoring · Decentralized coordination

1 Introduction

We aim to develop a monitoring system for decentralized and crowdsourced network infrastructures such as Guifi.net, a community network with more than 30,000 networking devices with IP addressing. The infrastructure of Guifi.net can be understood as a crowd-sourced, multi-tenant collection of heterogeneous network devices (wired and wireless) interconnected between them and forming a collective communication and computing system [1].

© Springer Nature Switzerland AG 2020
I. Brandic et al. (Eds.): ALGOCLOUD 2019, LNCS 12041, pp. 42–58, 2020.
https://doi.org/10.1007/978-3-030-58628-7_4

An edge cloud computing paradigm may initially be considered to build this monitoring system. In edge computing, cloud services running in data centers, e.g. data storage, are extended with the capacities of local processing at the edge. In such cloud-based services, which leverage edge devices, improved response time is achieved through local processing at the edge device and reduction of communication needs with remote data centers [2]. In the practical implementation of such a paradigm, the edge devices would be deployed as part of the vendor's monitoring application. They would need a suitable physical locations available in Guifi.net. In addition, the community network would need to assume the economic cost for the provision of the monitoring infrastructure.

Collaborative edge computing in Guifi.net started to be researched and developed in the last few years [3]. Currently in 2019 there are tenths of operational interconnected edge devices in the community network, which host diverse local services. These edge devices, such as mini-PCs or Single-Board-Computers (SBC), are located at the premises of some members of the community. Furthermore, the owners have system administrator permissions of the device, which enables them to install any required service. Differently, commercial edge gateways are typically locked and operate in a dedicated mode for the vendor's application. In fact, the edge computing model in community networks is radically different to the locked devices and application model of vendors: owners of edge devices in Guifi.net are encouraged to collaborate and actively contribute to the provision of network monitoring and end user-oriented services, and sustain edge micro-clouds [4,5].

The goal of the targeted monitoring system is to leverage the geographically-scattered low-capacity computing devices available at the premises of Guifi.net members to host the monitoring system software components. Since in the given context network partitions and failures of these servers may happen, and there is no traditional cloud data center infrastructure available in Guifi.net, we pursue a decentralized solution for the monitoring system. As a consequence, we envisioned to store monitoring data by means of a distributed replicated database. Furthermore, the software of the monitoring system will run as an additional service on the low-capacity computing devices that belong to Guifi.net members.

2 Needs for Monitoring System and Use Cases

The aim of a network monitoring system is to keep aware about the operational status of a network infrastructure, including the detection of anomalies and the collection of usage and performance data for every network component. We first describe the limitations of the current monitoring system, and we describe the requirements about economic compensation of network provision and usage.

2.1 Limitations of the Current Monitoring System

The current legacy monitoring system for the Guifi.net network aims at offering a public up-time and traffic accounting service. It consists of several independent, non-coordinated, crowd-sourced monitoring servers (built on several off-the-shelf Linux x86 low-end devices). Each server gets the list of nodes (i.e., network devices) to monitor from the Guifi.net website and, periodically, checks them for reachability/uptime (via ICMP pings) and measures network traffic (via SNMP). A limitation of this service is that each node in Guifi.net is assigned to only one monitoring instance/server. Therefore, if a monitor server fails, all the nodes under its supervision stop being monitored. Furthermore, monitoring data from a certain node is stored only by the assigned monitoring server, so data loss is prone to occur. Finally, the system is not self-adaptive to changes and requires manual intervention.

The current monitoring system impedes the implementation of relevant use cases, as sketched in the following subsection, and a more resilient, self-adaptive monitoring system is needed.

2.2 Needs of a Billing and Economic Compensations System

Currently, more than 20 companies operate their services professionally on top of the Guifi.net community network. For this, the project has put in practice a disruptive economic model [6] based on the commons and collaborative economy models, by means of the deployment of a common-pool network infrastructure and a fair and sustainable economic exploitation[1]. There is need for a comprehensive and reliable source of traffic data in the network.

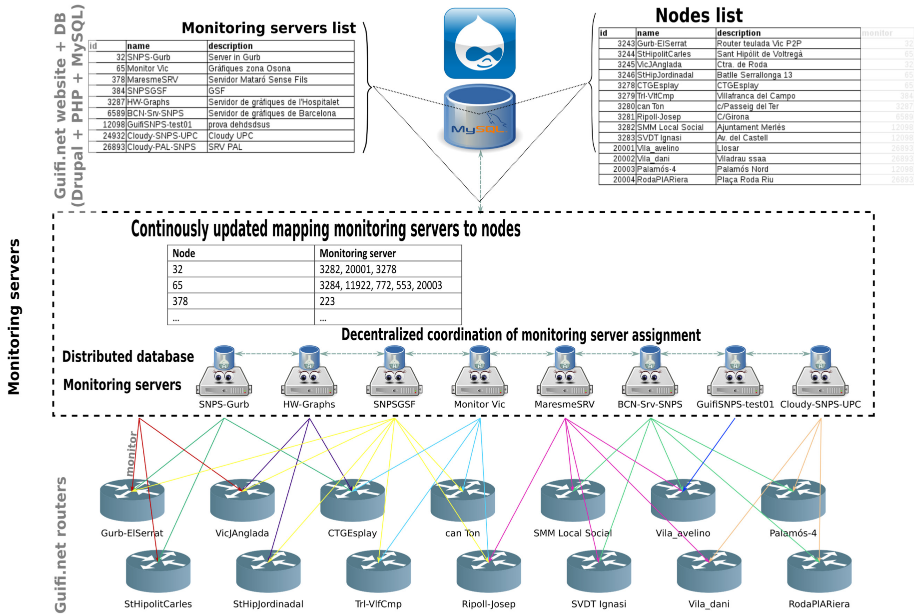
The current billing system to balance the community effort and the commercial activity related to contribution of network capacity and consumption of connectivity mostly relies on the traffic accounting being performed at the core routers by the Guifi.net Foundation. However, this requires a considerable amount of manual intervention to combine, verify and validate not only the figures, but also the data reported by the ISPs themselves. An automated billing mechanism enabled by the retrieval of reliable monitoring information from a shared data layer would help in making the operation of the network more sustainable, both technically and economically. In addition, publicly-available traffic and resources usage data facilitated by a monitoring system would improve the transparency for the whole ecosystem, leading to less disputes between the Guifi.net Foundation and the ISPs.

3 An Edge Monitoring System for Network Infrastructures

The new monitoring system aims at solving the limitations of the legacy monitoring system and provide comprehensive and reliable monitoring data for all

[1] Guifi.net - The economic project - https://guifi.net/en/economic-project.

network devices despite network and server failures. We describe the system architecture and its implementation.



id	name	description
32	SNPS-Gurb	Server in Gurb
65	Monitor Vic	Gràfiques zona Osona
378	MaresmeSRV	Servidor Mataró Sense Fils
384	SNPSGSF	GSF
3287	HW-Graphs	Servidor de gràfiques de l'Hospitalet
6589	BCN-Srv-SNPS	Servidor de gràfiques de Barcelona
12098	GuifiSNPS-test01	prova dehdsdsus
24932	Cloudy-SNPS-UPC	Cloudy UPC
26893	Cloudy-PAL-SNPS	SRV PAL

id	name	description	monitor
3243	Gurb-ElSerrat	Router teulada Vic P2P	32
3244	StHipolitCarles	Sant Hipòlit de Voltregà	65
3245	VicJAnglada	Ctra. de Roda	32
3246	StHipJordinadal	Batlle Serrallonga 13	65
3278	CTGEsplay	CTGEsplay	65
3279	Trl-VlfCmp	Vilafranca del Campo	384
3280	can Ton	c/Passeig del Ter	3287
3281	Ripoll-Josep	C/Girona	6589
3282	SMM Local Social	Ajuntament Merlès	12098
3283	SVDT Ignasi	Av. del Castell	12098
20001	Vila_avelino	Llosar	26893
20002	Vila_dani	Viladrau ssaa	26893
20003	Palamós-4	Palamós Nord	12098
20004	RodaPlARiera	Plaça Roda Riu	26893

Node	Monitoring server
32	3282, 20001, 3278
65	3284, 11922, 772, 553, 20003
378	223
...	...

Fig. 1. Architecture of the monitoring system showing the different components.

3.1 Architecture

The architecture of the monitoring system, showing the monitoring servers and their integration with the Guifi.net website and network nodes, is depicted in Fig. 1. On the top of the picture appears the Guifi.net website and its central database, 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 available at any time. Compared to the current implementation (i.e., the legacy monitoring system) with a fixed mapping of network nodes to monitoring servers, the new monitoring system proposes a mapping, that is decentralized and autonomous, managed by the monitoring servers themselves, and dynamically updated to ensure sufficient coverage and balance in the monitoring.

Considering Algorithm 1, once the monitoring servers know the list of nodes to watch (Phase 1 - Monitor registration), they coordinate with each other (Phase 2 - Self-assignment) indirectly over the mutable data object given by the *monitoring servers* ⇔ *network devices* mapping in order to perform the actual monitoring of all nodes. The objective is to assign every single network

Algorithm 1 Assign algorithm with policies, for node *id*

```
Require:
    dbhost                          ▷ AntidoteDB hostname/IP address
    dbPort                          ▷ AntidoteDB TCP port
    id                              ▷ Unique ID of the monitor in the network
    minMonitors                     ▷ Min # of monitors a device needs
    maxMonitors                     ▷ Max # of monitors a device needs
    maxDevices                      ▷ Max # of devices the monitor can watch
    policy    ▷ Network devices to monitoring servers assignation policy (1, 2, 3)
    Phase 1 – Monitor registration
 1: procedure Registration(G)
 2:     monitorsList[] ← GetGlobalMonitorsList
 3:     AddMonitorToList(id, monitorsList[])
 4:     UpdateGlobalMonitorList(monitorsList[])
 5: end procedure
    Phase 2 – Monitor Self-assignment
 6: procedure Assign(id)
 7:     numDevices ← 0
 8:     devicesInAntidote[] ← getDevicesInAntidote()
 9:     for each device in devicesInAntidote[] do
10:         if (id is in device.monitors[]) then
11:             numDevices++
12:         end if
13:     end for            ▷ We find out the total numDevices this node monitors
14:     switch policy do
15:         case 1 ▷ Min: Assign if network device not enough covered (minMonitors)
16:             for each device in devicesInAntidote[] do
17:                 if (sizeOf(device.monitors[]) < minMons)
                       && (numDevices < maxDevices)          then
18:                     assignMonitorToDevice(id, device), numDevices++
19:                 end if
20:             end for
21:         case 2  ▷ Max: Assign if network device not fully covered (maxMonitors)
22:             for each device in devicesInAntidote[] do
23:                 if (sizeOf(device.monitors[]) < maxMons)
                       && (numDevices < maxDevices)          then
24:                     assignMonitorToDevice(id, device), numDevices++
25:                 end if
26:             end for
27:         case 3   ▷ Fair: Assign if device not well covered ([minMons, maxMons])
28:             maxMons ← minMons+1
29:             for each device in devicesInAntidote[] do     ▷ order by asc #mons
30:                 if (sizeOf(device.monitors[]) ∈ [minMons, maxMons])
                       && (numDevices < maxDevices)                   then
31:                     assignMonitorToDevice(id, device), numDevices++
32:                 end if
33:             end for
34: end procedure
```

device to –at least– a minimum number of monitoring servers. This task can be performed in different ways. Currently, three policies are implemented. Each policy leads to different properties of the monitoring system, elaborated in the following Sect. 4.

The data manipulated by the monitoring system draws from two sets of immutable objects and creates a mapping between these objects. The first set contains a list with all the network devices in Guifi.net that have to be monitored. The list of network devices in the whole Guifi.net contains more than 30,000 nodes. 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 inside the network, with its own IP address, etc. that must be monitored too). The data in this second set is only modified by authoritative updates issued from the Guifi.net website; again, the monitoring servers only read it but do not modify it.

In the *assign* operation (Phase 2 – Monitor Self-assignment), any monitoring server may modify the mapping between network devices and monitoring servers (add, update or remove these relations at any time).

The assignment in the *monitoring servers* ⇔ *network devices* mapping will change over time, as new network devices are added to the list, the network conditions change, workload is redistributed, monitoring servers join or exit the pool, etc. As a consequence, each monitoring server continuously and concurrently –not in synchronisation with the other servers– reads and writes to the shared distributed mapping object.

After conducting the *monitor self-assignment* procedure in Algorithm 1, in Algorithm 2 additional operations take place. Depending on policy, if the number of monitors for a device exceeds the requirement, a monitor may un-assign itself from the list of monitors of a device. With regards to un-assigning other unresponsive monitors, the sanitize function is performed. By means of outdated timestamps, disconnected monitors are detected and an active monitor erases them from the *monitoring servers* ⇔ *network devices* mapping. In case of a controlled disconnection of a monitor, it performs the de-registration function, in which a monitor un-assigns itself from the list of monitored devices before un-assigning itself from the list of available monitors.

In order to successfully deal with the required concurrent updates of the *monitoring servers* ⇔ *network devices* mapping shared among all monitors, the data consistency and integrity between the different database instances is needed. If these properties are kept, it can be ensured that all network nodes eventually end up being properly assigned to monitoring servers.

3.2 Implementation

We developed the monitoring system as a prototype implementation that uses the Go language. The system is composed of four components to conduct the

Algorithm 2 Algorithm for un-assign, sanitization and de-registration

Phase 1 – Monitor Self-unassignment

```
1: procedure UNASSIGN(id)
2:     devices[] ← getDevicesInAntidote()
3:     for each device in devices[] do
4:         if device.monitors[] > minMonitors then
5:             unassignMonitorFromDevice(id, device)
6:         end if
7:     end for
8: end procedure
```

Phase 2 – Global assignment sanitization

```
9: procedure SANITIZE
10:     monitorsList[] ← GetGlobalMonitorsList
11:     devices[] ← getDevicesInAntidote()
12:     for each device in devicesInAntidote[] do
13:         for each monitor in device.monitors[] do
14:             if (monitor is not in monitorsList[]) then
15:                 unassignMonitorFromDevice(monitor,
                        device)
16:             end if
17:         end for
18:     end for
19: end procedure
```

Phase 3 – Monitor Self-deregistration

```
20: procedure DEREGISTRATION(id)
21:     monitorsList[] ← GetGlobalMonitorsList
22:     RemoveMonitorFromList(id, monitorsList[])
23:     UpdateGlobalMonitorList(monitorsList[])
24: end procedure
```

operations named fetch, assign, ping and snmp. The source code is available at our GitLab repository[2]. For the consistency of the data in the distributed database instances, the AntidoteDB database was chosen [7]. AntidoteDB implements Conflict-Free Replicated Data Types (CRDT), which offer strong eventual consistency [8]. The integration between the monitoring server component and AntidoteDB is done through AntidoteDB's Go client.[3]

4 Evaluation of the Assign Algorithm

Our objective is to study the *assign* component, specifically the algorithm of the assign operation, when using three different policies for assigning the monitors to network devices. For this purpose, we aim to observe the evolution of

[2] https://lightkone.guifi.net/lightkone.

[3] https://github.com/AntidoteDB/antidote-go-client.

the assignment of network devices, when monitoring servers perform the *assign* operation join and leave.

For conducting the experiments, first, using the *fetch* component of the monitoring system, a data file with 54 devices of a small region of Guifi.net infrastructure is stored in to the AntidoteDB storage service in order to have it for the monitors network devices to be assigned to these servers. For observing the *assign* operation in the experiments, a customized setting with shorter routine execution periods of 10 s is configured (instead of the default value of 150 s). The required minimum number of monitors per network device ($minMonitors$) is set to 3 monitors. The maximum number of network devices per monitor ($maxDevices$) is set to 50 devices. The assignment state is dumped every 5 s, which is half of the period of the assign operations. In the first half of the experiment duration, assign clients join one by one the monitoring system. In the second half of the experiment duration, the assign clients gradually leave the monitoring system.

The experiments for policy 1 and 2 are undertaken with a local AntidoteDB instance, to which during the experiment up to 10 assign client write. The experiment for policy 3 is conducted with 8 Minix devices and 7 assign clients in Guifi.net.

4.1 Assignment Policy 1 (min): Reach $minMons$ per Network Device

In policy 1 each monitoring server self-assigns those network devices that have less monitoring devices assigned than the minimum number specified by the $minMons$ parameter. Devices are picked randomly (no preference) until the server reaches its maximum number of monitored devices, specified by the $maxDevices$ parameter of the monitor.

In this experiment 10 assign clients join and leave the monitoring system during 7 min. In the first 3 min the assign servers join every 20 s, one by one, and after approximately 4 min, they gradually leave until having 0 clients at the end of the experiment.

Figure 2 shows the assigned devices with policy 1. The configuration has set the parameters $minMons = 3$, which requires a minimum of 3 monitors per device. With 54 devices in the dataset, the number of assigned devices must reach 162. Before 100 s, this number of assigned devices is reached. With a monitoring capacity configured as 50 devices per monitor, which 4 monitors the number of 162 assigned devices is reached. After 4 min, *one of assigned monitors* disconnects. When the other idle monitors periodically connect (unsynchronized with each other) to the system, they detect the under-monitoring of the system, and as a consequence, self-assign devices to be monitored as to their monitoring capacity ($maxDev = 50$). Since these operations are done concurrently among several monitors and while the local decision is not updated in the shared *monitoring servers* $\Leftrightarrow$ *network devices* mapping, the number of assigned devices raises. As the monitors get disconnected, down to 0 monitors, the number of assigned devices decreases correspondingly.

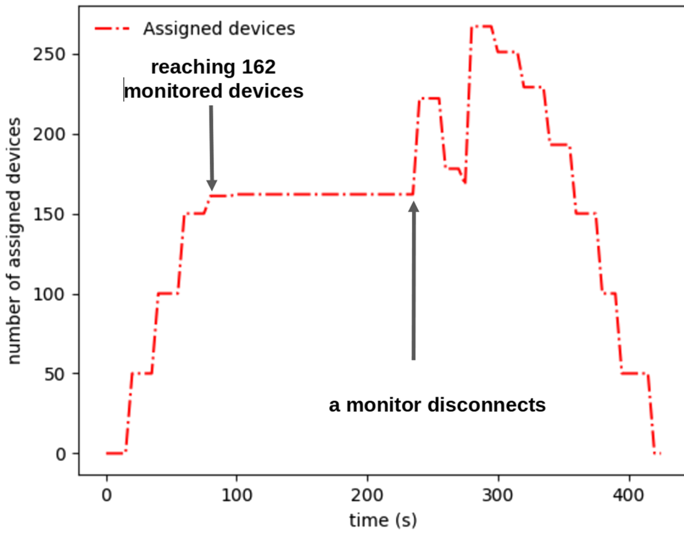


Fig. 2. Assigned devices with policy 1.

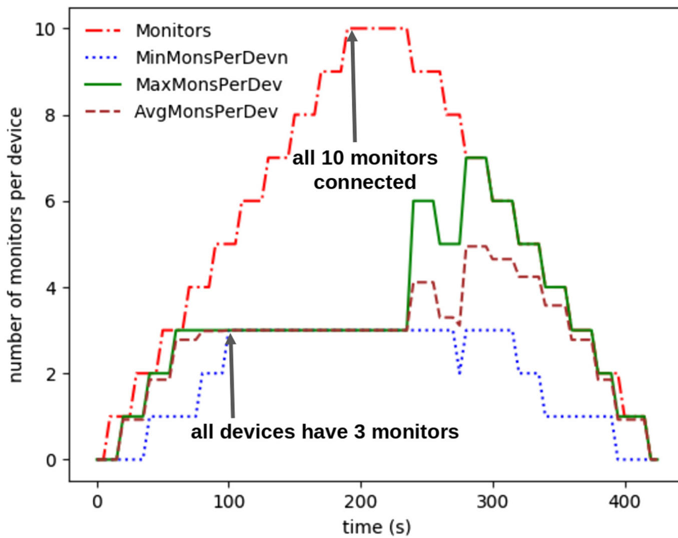


Fig. 3. Monitors per devices with policy 1.

Figure 3 shows the monitors per device with policy 1. It can be seen that after around 100 s, all devices have at least 3 monitors. This assignment corresponds to the configured system requirements and remains stable until the disconnection of one of the assigned monitors starts at minute 4. It can be seen that at some instant the maximum number of monitors per device (MaxMonPerDev) raises

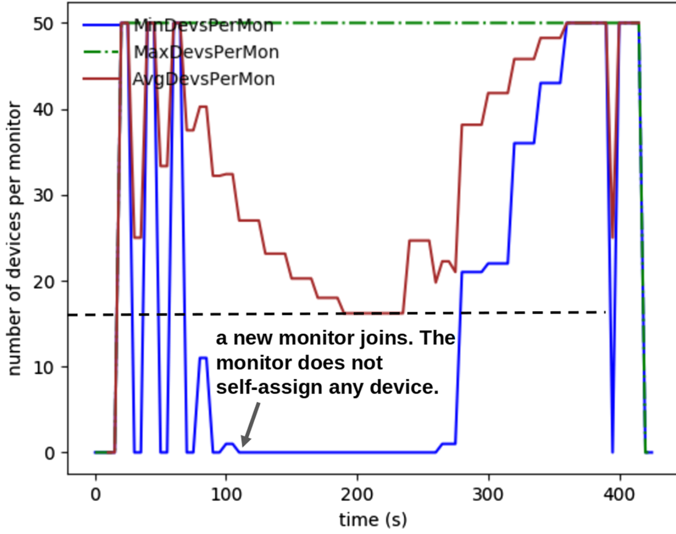


Fig. 4. Devices per monitor with policy 1.

up to 7, which corresponds to the concurrent responses of the idle monitors to take over the monitoring duties of the disconnected monitors.

Figure 4 shows the devices per monitor with policy 1. It can be seen that after around 100 s the value for the minimum number of devices per monitor stays at 0 during some time of the experiment. This can be explained since new joining monitors detect that the system requirements are satisfied and remain in idle state, without self-assigning any new devices to be monitored.

4.2 Assignment Policy 2 (max): Reach *maxMons* per Network Device

In policy 2 the system uses *all* the possible monitoring resources. Devices are picked starting by the ones with the least monitors.

Figure 5 shows the sum of assigned devices with policy 2. Different to policy 1, policy 2 ignores any minimum number of monitors per device and the monitors self-assign devices to be monitored up to reaching its maximum monitoring capacity. It can be seen that a sum of up to 500 assigned devices is reached, which corresponds to the 10 monitoring servers and a capacity of 50 devices each to monitor.

Figure 6 shows the monitors per device with policy 2. Since all available monitors are assigned, devices obtain up to 10 monitors shortly before 200 s of the experiment, which corresponds to all 10 monitors connected.

In Fig. 7 it can be seen that at around 200 s the value for the minimum number of devices per monitor reaches 50, which corresponds to the fact that all

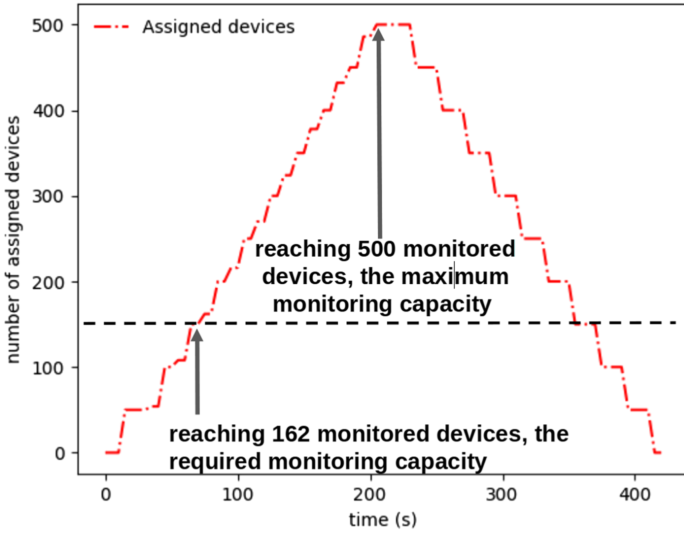


Fig. 5. Assigned devices with policy 2.

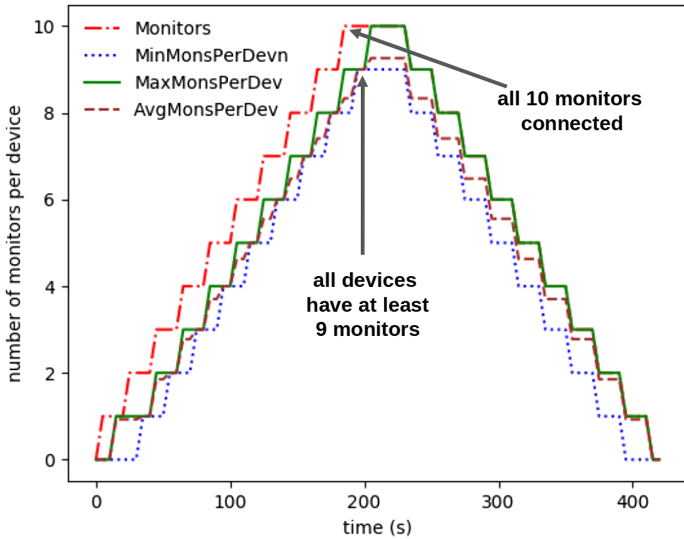


Fig. 6. Monitors per devices with policy 2.

joined monitors operate at the maximum monitoring capacity, corresponding to the behaviour expected from policy 2.

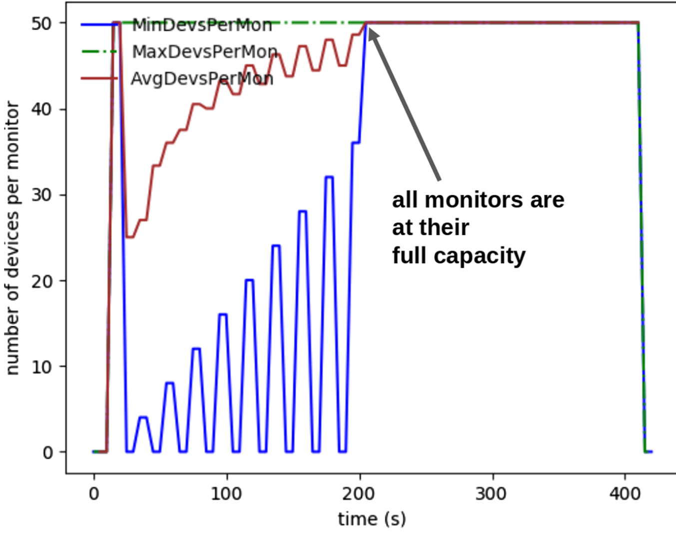


Fig. 7. Devices per monitor with policy 2.

4.3 Assignment Policy 3 (fair): Reach $[minMons, maxMons]$ per Network Device

In policy 3, devices are picked starting by the ones with the least monitors until they are monitored by $minMons$ monitors. If there are still monitoring

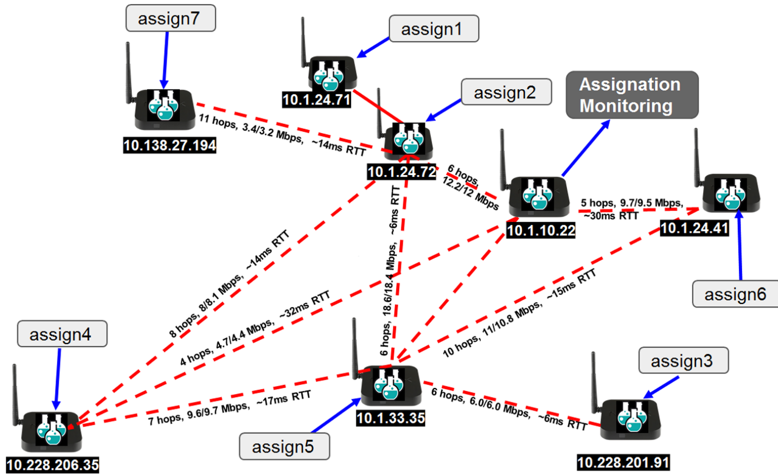


Fig. 8. Testbed for the assign operation with policy 3. The testbed is deployed in the GuifiSants wireless mesh network (see footnote 4).

capacities available, devices will be monitored by at most $maxMons$ (by default, $maxMons = minMons + 1$).

We conduct this evaluation by deploying the monitoring system on real nodes of Guifi.net. In the Guifi.net environment, monitoring servers consist of different hardware, which can range from resource-constraint SBCs to desktop computers. In order to represent this situation, we have have installed several x86 mini-PCs in a wireless mesh network part of Guifi.net (at users' homes) to form a testbed in which these devices operate as monitoring servers[4].

Figure 8 illustrates the deployed testbed and provides some information about the network characteristics (IP, bandwidth between nodes and RTT). The eight black nodes correspond to Minix devices (Intel Atom x5-Z8350 4-cores CPU @ 1.44 GHz, 4 GB of DDR3L RAM and 32 GB eMMC) running Debian Strech. Each Minix device hosts an AntidoteDB instance. Most of the Minix devices are geographically far from each other with a few hops of wireless links between them. As shown in the Fig. 8, we use 7 assign clients, which are hosted on the same Minix device they write to. On the $8th$ Minix device, we installed a component which reads the current assignments from the AntidoteDB instance every 5 s. Note that all 8 AntidoteDB instances are fully replicating the data.

Figure 9 shows the assigned devices with policy 3. The configuration has set the parameters $maxMons = minMons + 1$ ($minMons = 3$), which requires a minimum of 3 monitors per device, and 4 monitors if there are available monitoring capabilities in the system. With 54 devices in the dataset, the number of assigned devices can reach up to 216 for 4 monitors per device. At around 100 s, this capacity is reached. With a monitoring capacity configure as 50 devices per monitor, which 4 monitors the number of 162 assigned devices is reached, requiring 5 monitors for 216 assigned devices. After 4 min, *one of assigned monitors* disconnects. When the idle monitors periodically connect, they detect the under-monitoring, and as a consequence, self-assign devices as to their monitoring capacity ($maxDev = 50$). Since this operation is done concurrently among several monitors, the number of assigned devices raises. The local decisions are then communicated to the shared *monitoring servers* $\Leftrightarrow$ *network devices* mapping to allow coordination in the next periodic assign operation of each monitor. As in the second half of the experiment the monitors become disconnected down to 0 monitors, the number or assigned devices decreases correspondingly.

4.4 Comparison of Policies

Table 1 compares the three assignment policies. Each of the policies target to achieve a certain property of the monitoring system. In policy 1 the joining monitors self-assign devices up to their maximum monitoring capacity. Once all devices are monitored by the minimum number of monitors, no additional assignment takes place. It can be considered that policy 1 is resource consumption efficient, by having the least number of monitors doing active monitoring at

[4] The wireless mesh network is GuifiSants; nodes and network topology can be found at http://dsg.ac.upc.edu/qmpsu/index.php.

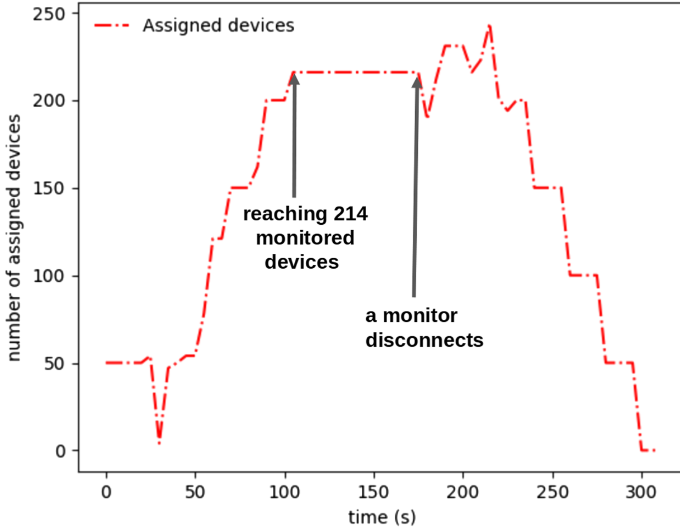


Fig. 9. Assigned devices with policy 3.

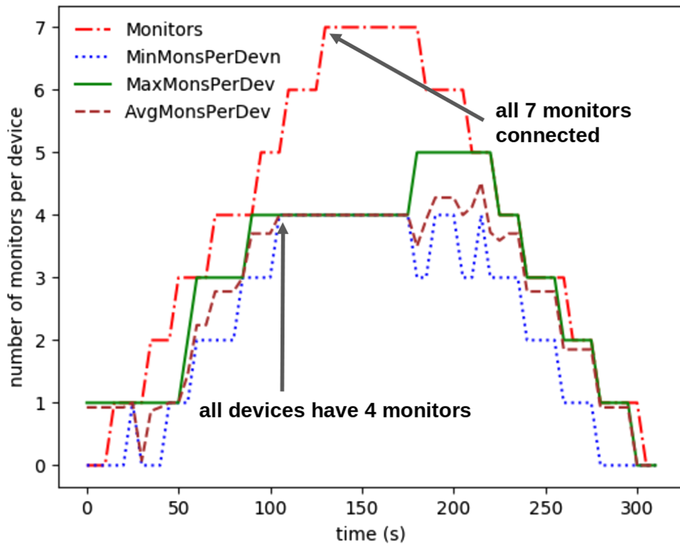


Fig. 10. Monitors per devices with policy 3.

their maximum capacity. In policy 2, the maximum redundancy for the monitoring task is pursued by assigning the total monitoring capacity of the monitors, even if the system requirements for the minimum number of monitors per device are exceeded. By activating all monitors, policy 2 is less resource consumption friendly. Policy 3 implements a trade-off between policy 1 and 2. By setting the

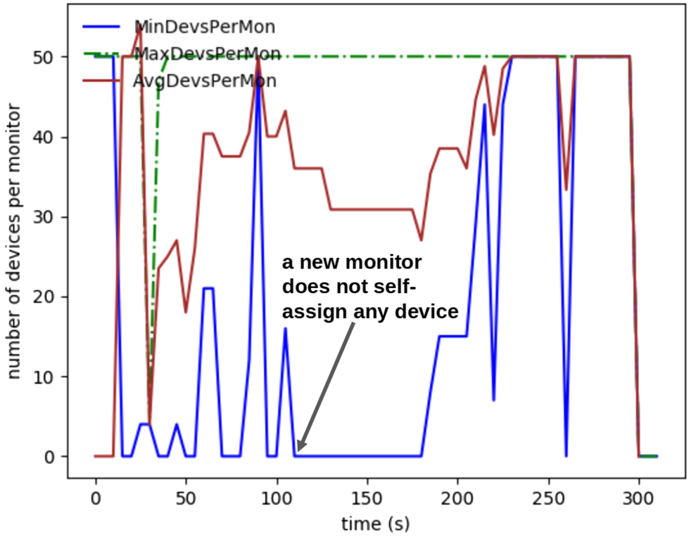


Fig. 11. Devices per monitor with policy 3.

Table 1. Comparison of the three assignment policies.

Policy	Pros	Cons
1	Once the system requirement are reached, additional monitors remain idle	In some corner cases, depending on the timing the servers joined the system, some devices could end up not being monitored even if there are enough resources available
2	Every device is overwatched. Increased redundancy	The more monitoring servers are available, the more waste of resources. No idle monitors
3	Exceeding slightly monitoring requirements if servers become available	Possibly idle monitors, not maximum redundancy

maxMons parameter, the minimum system requirement can be exceeded slightly, while if there is an excess of monitoring capacity available beyond the maxMons requirement, it will remain idle (Fig. 10).

5 Related Work

In the presented monitoring system, we focus on the control algorithm, how the distributed local monitors coordinate among them their actions. We consider that decisions are taken in a decentralized way by each local monitor, based on the information of the global state retrieved periodically from the distributed

database, instead of retrieving the order for its monitoring actions from a centralized controller. We do not focus on the actual monitoring of network data, which corresponds to the actions of the monitors once the assignment is done. Differently, in [9] the lack of a centralized data storage is motivating a decentralized monitoring approach, where monitoring data is communicated among local monitors, which face the challenge of taking global decisions based on an incomplete view about the global monitoring data (Fig. 11).

In [10], targeting large-scale Software-Defined Networks (SDN), multiple monitoring entities that perform monitoring tasks autonomously and without maintaining a global view of the network run-time state are proposed. The goal is to enable these local managers to adaptively reconfigure the network resources under their scope of responsibility. The local managers, integrated in a so called monitoring module (MM), communicate northbound with a higher level management applications (MAs), and southbound with the SDN controller. The MAs change the monitoring requirements for the MMs. Differently, in our work there is no equivalent to the MA. Our monitors coordinate horizontally (over the shared database mapping object) with the other monitors their actions. The requirements for the monitoring that each monitor does are not obtained from a MA, but determined by each monitor from the actions of the other monitors.

Graphite [11] is monitoring tool able to run on low-cost hardware and cloud data centres. Graphite integrates with several other professional tools for data storage and visualization. While Graphite allows for many customization options, this flexibility is applied to the actual monitoring task. Context awareness to take into account in deciding about the monitoring operation, individual automatic re-configurations in response to a current situation in the network or monitoring system itself, seem to be less well supported.

Many other time series databases [12] exhibit centralization and the intrinsic problem of distributing writes while keeping full consistency, that depends on a data storage layer that ensures strong eventual consistency. This is the differentiating aspect of the monitoring system presented in this work, in the use of CRDT-based data structures to enable strong eventual consistency of the mapping object, which is used to by each monitor to decide on its individual monitoring duties, with concurrent writes to shared data structures.

6 Conclusions

A distributed monitoring system for a crowdsourced network infrastructure was presented. Situated in a dynamic edge environment in which failures and network partitions may happen, a decentralized approach was proposed to built the monitoring system. A key design was the shared distributed *monitoring servers* ⇔ *network devices* mapping to allow the coordination between the periodic assign operations of each monitor. The monitoring system was implemented and leverages the AntidoteDB database, which provides a strong eventually consistent data storage service for distributed replicas of data. The assign operation, decentrally conducted by all monitoring servers while they coordinate over

a shared distributed data object, was evaluated. The functional performance of three assignment policies was analyzed. The different policies are interpreted from the point of view of the trade-off between resource consumption and redundancy, effects to be taken into account when determining the monitoring system requirements. Future work can consider enhancements of the assign policies, by becoming more context-aware, for instance by each monitor taking into account network conditions and network device attributes.

Acknowledgment. This work was supported by the European H2020 framework programme project LightKone (H2020-732505), by the Spanish government contract TIN2016-77836-C2-2-R and PID2019-106774RB-C21 by the Catalan government contract AGAUR SGR 990.

References

1. Baig, R., Roca, R., Freitag, F., Navarro, L.: Guifi.net, a crowdsourced network infrastructure held in common. Comput. Netw. **90**, 150–165 (2015)
2. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017)
3. Baig, R., Freitag, F., Navarro, L.: Cloudy in guifi.net: establishing and sustaining a community cloud as open commons. Future Gener. Comput. Syst. **87**, 868–887 (2018)
4. Selimi, M., Cerdà-Alabern, L., Freitag, F., Veiga, L., Sathiaseelan, A., Crowcroft, J.: A lightweight service placement approach for community network micro-clouds. J. Grid Comput. **17**(1), 169–189 (2019)
5. Sathiaseelan, A., et al.: Towards decentralised resilient community clouds. In: Proceedings of the 2nd Workshop on Middleware for Edge Clouds & Cloudlets, MECC 2017, pp. 4:1–4:6. ACM, New York (2017)
6. Cerdà-Alabern, L., Baig, R., Navarro, L.: On the guifi.net community network economics. Comput. Netw. **168**, 107067 (2020)
7. AntidoteDB: A planet scale, highly available, transactional database (2019). https://www.antidotedb.eu/
8. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 386–400. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24550-3_29
9. Falcone, Y., Cornebize, T., Fernandez, J.-C.: Efficient and generalized decentralized monitoring of regular languages. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 66–83. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43613-4_5
10. Tangari, G., Tuncer, D., Charalambides, M., Pavlou, G.: Decentralized monitoring for large-scale software-defined networks. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 289–297, May 2017
11. Davis, C., contributors: Graphite: an enterprise-ready monitoring tool. https://graphiteapp.org/
12. Bader, A., Kopp, O., Falkenthal, M.: Survey and comparison of open source time series databases. Datenbanksysteme für Business, Technologie und Web (BTW 2017)-Workshopband (2017)