Article

Context Diffusion in Fog Colonies: Exploring Autonomous Fog Node Operation Using ECTORAS

Vasileios Nikolopoulos , Mara Nikolaidou * , Maria Voreakou and Dimosthenis Anagnostopoulos

Department of Informatics and Telematics, Harokopio University of Athens, 17778 Tavros, Greece; basil@hua.gr (V.N.); voreakou@hua.gr (M.V.); dimosthe@hua.gr (D.A.)

* Correspondence: mara@hua.gr

Abstract: In Fog Computing, fog colonies are formed by nodes cooperating to provide services to end-users. To enable efficient operation and seamless scalability of fog colonies, decentralized control over participating nodes should be promoted. In such cases, autonomous Fog Nodes operate independently, sharing the context in which all colony members provide their services. In the paper, we explore different techniques of context diffusion and knowledge sharing between autonomous Fog Nodes within a fog colony, using ECTORAS, a publish/subscribe protocol. With ECTORAS, nodes become actively aware of their operating context, share contextual information and exchange operational policies to achieve self-configuration, self-adaptation and context awareness in an intelligent manner. Two different ECTORAS implementations are studied, one offering centralized control with the existence of a message broker, to manage colony participants and available topics, and one fully decentralized, catering to the erratic topology that Fog Computing may produce. The two schemes are tested as the Fog Colony size is expanding in terms of performance and energy consumption, in a prototype implementation based on Raspberry Pi nodes for smart building management.

Keywords: Fog Colony Control; autonomous nodes; context awareness; publish/subscribe protocols

Citation: Nikolopoulos, V.; Nikolaidou, M.; Voreakou, M.; Anagnostopoulos, D. Context Diffusion in Fog Colonies: Exploring Autonomous Fog Node Operation Using ECTORAS. *IoT* **2022**, *3*, 91–108. <https://doi.org/10.3390/iot3010005>

Academic Editor: Marco Picone

Received: 14 November 2021

Accepted: 14 January 2022

Published: 18 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

During the last years, a lot of research has been targeted around Fog and Cloud computing. As it was first defined by Cisco in [1], Fog Computing is a system that can transfer data with ease in a wireless and distributed environment, conducting near-field communication with sensors [2]. As defined in [3], Fog Computing provides the quality of service required by certain smart applications, such as traffic control, health care applications, and autonomous driving. Although Fog is different from Cloud, the benefits of Cloud should be preserved with these extensions to Fog, including containerization, virtualization, orchestration, manageability, and efficiency [4–8].

Recently, the Fog Colony notion was introduced [9–12]. Fog colonies, as described in [9], are small-scale data centers made up from an arbitrary number of Fog Nodes. The importance of the Fog Colony is based on the data sharing as well as the services' distribution among the Fog Nodes. A Fog Colony could be defined as a parent container to Fog Nodes based on network topology, physical topology, or applied operational policies [13].

A Fog Colony may act either as a centralized or a decentralized system. In centralized systems, there is at least one control entity that manages the Fog Colony [14–17]. The definition of this entity can differ from approach to approach depending on the level of administration rights it possesses. Common terminology used in the literature regarding the control node is fog control node, fog master node, orchestrator node, etc.

In [3], the authors presented a centralized Fog Colony solution, with a master fog node which schedules to forward the offloaded tasks to available Fog Nodes. The importance of this research was based on the ant colony optimization of the task-offloading. Authors of [18] propose a similar approach to [3], as they use a fog orchestrator node which is responsible for the smooth and operational communication between the Fog Nodes.

In [19], the authors propose a Fog Colony architecture where Fog Nodes are able to exchange knowledge information. The centralized entity in this case, called Registry Unit, acts as a service registry repository and partly as a message broker. The Fog Nodes are deemed autonomous, as they possess a number of properties that allow them to manage themselves. In [20], the use of a DDS-based protocol [21] for information exchange between autonomous Fog Nodes is proposed. The protocol is based on publish/subscribe logic and uses topics for allowing the cooperation of the nodes [22,23]. To allow Fog Nodes to exchange information on a specific subject, represented as a discrete topic, they have to be registered in a broker-like centralized unit, the Registry Unit, while supported topics (e.g., categories of information to be shared) are also controlled by the Registry Unit.

In this work, we aim to explore different techniques for context diffusion within Fog Colonies, with our ultimate goal being the autonomous operation of Fog Nodes. To do so, different methods of communication between the Fog Nodes are explored. To that end, we extend the functionality of the information exchange protocol introduced in [20], and further explore the Fog Colony architecture by proposing two alternative modes of operation, one supporting a broker and one fully decentralized. To enable comparison of the two suggested modes of operation, a number of performance tests were executed. Colonies of different sizes, running the two different flavors of the implemented protocol, were set up and the effect of the two different implementations on the performance of the Fog Colony was measured.

The rest of the paper is structured as follows: Section 2 discusses work related to Fog colonies and collaboration mechanisms between Fog Nodes. Section 3 presents the Context Aware Autonomous fog node concept and discusses its internal architecture. Section 4 presents in detail the information exchange protocol we will be implementing and discusses the expected benefits and drawbacks of its two different implementations. Section 5 presents the findings from the execution of our performance tests, while Section 6 discusses the findings from Section 5. Conclusions and future work reside in Section 7.

2. Related Work

As mentioned in the introduction, this work aims towards exploring different methods of diffusing context, ultimately achieving autonomous operation of Fog Nodes within Fog Colonies. To that end, in this chapter, we will make a brief introduction into some of the most important Fog Computing terms and concepts related to this work. Additionally, we will present various already existing mechanisms for exchanging information within Fog Colonies which aim to solve similar challenges.

2.1. Related Work and Terminology

2.1.1. The Fog Ecosystem

When discussing about the high-level architecture of the Fog ecosystem, the most common representation is a layered approach [11,12,24–27], as seen in Figure 1. As defined by the OpenFog Consortium [4] “Fog Computing is an extension of the traditional Cloud-based computing model where implementations of the architecture can reside in multiple layers of a network’s topology. However, all the benefits of the Cloud should be preserved with these extensions to Fog, including containerization, virtualization, orchestration, manageability, and efficiency”.

The main entities that live in a Fog Computing environment are the Fog Nodes, communicating with entities from the same or different layers. In Figure 1, we can see an optical representation of the world that exists around Fog Nodes. Starting from the top of the figure, we have the Cloud Layer. In Cloud Layer, we group all services that are cloud-based. For the Fog Nodes, the Cloud is another service they can reach when needed, but is far away from them. After the Cloud Layer, we have the Fog Layer. Within the Fog Layer, we have groups of cooperating Fog Nodes, named Fog Colonies. Each Fog Colony is composed of any number of Fog Nodes. Below the Fog Layer, we have the User Domain. In the User Domain, we group every service, device, and entity that regards the end user

of the services provided by the Fog Layer. Finally, below the User Domain, we have the WSN layer. Within the WSN layer, any number of different WSNs can exist. Fog Nodes are usually tasked with communicating with WSNs as an interface and data pre-processor between the IoT and the Cloud.

Figure 1. Fog Nodes can communicate with multiple and different actors at the same time.

2.1.2. Fog Node Collaboration

The most common challenges tackled by the literature regard transmission delays and scalability. Authors of [28] introduce a general framework for IoT-fog-cloud applications, and propose a delay-minimizing collaboration and offloading policy for fog-capable devices. In essence, their work revolves around offloading of tasks to neighboring Fog Nodes by implementing a heuristic approach that finds the best neighbor to offload a task.

Similarly to the above, authors of [29] exchange Fog Nodes' managed resources, such as sensors, to neighboring Fog Nodes. However, the work does not define how the nodes communicate and select possible new hosts for their resources.

Contrary to offloading to neighboring Fog Nodes, authors of [30] propose a different approach for minimizing transmission delay. In their paper, they propose a formal way of determining the optimum number of nodes that should be upgraded to Fog Nodes with additional computing capabilities, in order to maximize the average data rate, in a heterogeneous environment where not every node has the capabilities of a fog node.

2.1.3. Fog Colony Control

Another common discussion topic around Fog Colonies regards the Fog Colony Control. The work in [31] proposes the existence of colony-wide control software run by specific Fog Orchestration Control nodes. Similarly, in [11,27], Fog Controllers are introduced. An alternative to centralized control is seen in [32], where a flat model of structure is implemented with self-organizing compute nodes, similar to the nodes and architecture of [19].

2.2. Context Diffusion

2.2.1. Definition of Context

Context plays a significant role in Fog Node Collaboration. Defined in [33], "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". As discussed in [34], context awareness means that the context information is taken from the environment or provided by a client, and is exploited by a system to improve its overall quality. To that extent, it

is critical for the Fog Nodes to be contextually aware given the aim of collaboration. An extension of being contextually aware is that the Fog Nodes can further exploit the context information to make autonomous decisions without relying on centralized control.

In [35], the authors propose a Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing. In it, a three-tier software architecture is presented, split in the Cloud, the Fog, and the Edge layers. Although the notion of context and situational information is proposed, as well as the internal architectures of Fog and Edge nodes is discussed, the actual protocol to exchange this information is not discussed.

2.2.2. Context Diffusion Protocols

In [8], a Survey of various IoT- Fog-Cloud Ecosystems is presented. In it, most systems exploit the use of already existing protocols, such as CoAP, MQTT, and AMQP.

CoAP, or the Constrained Application Protocol, is a transfer protocol for constrained nodes and networks, such as those that exist in the IoT [7]. CoAP is similar to HTTP and uses the REST architectural style, based on UDP. In essence, CoAP acts as a peer to peer protocol, just like HTTP, but brings reduced overhead. A central element of CoAP is the reduced complexity that is brought by removing the TCP elements and using UDP. CoAP uses binary headers and compacts headers up to 20 bytes.

MQTT is one of the oldest Machine to Machine (M2M) communication protocols, initially introduced in 1999 [36]. It is a publish/subscribe messaging protocol designed for lightweight M2M communication in constrained networks. MQTT clients publish messages to an MQTT broker, which they are subscribed to by other clients, and messages may be retained for future subscribers. Every message is published to a relevant topic, in essence a specific address. MQTT uses TCP as a transport protocol and TLS/SSL for security.

AMQP is a lightweight M2M protocol initially introduced in 2003 [36]. AMQP supports both request/response and publish/subscribe architectures. It offers a wide range of features, such as reliable queueing, topic-based publish/subscribe messaging, flexible routing and transactions. Similarly with MQTT, it makes use of a broker to allow for publish/subscribe communications. AMQP exchanges messages in various ways, including in fanout form, by topic, or based on headers. It is a binary protocol and requires a fixed header of 8-bytes, with small message payloads, with the maximum size dependent on the broker. AMQP, similarly with MQTT, uses TCP as the default transport protocol and TLS/SSL and SASL for security.

3. Achieving Fog Node Autonomous Function

To investigate context diffusion in Fog Colonies and reach autonomous operation for Fog Nodes, it is important to discuss the functional requirements of a Fog Node to achieve this goal. In this chapter, the internal architecture of a Fog Node is explored to diffuse Context in the Fog Colony, understand received Context and exploit this context to achieve autonomous function. Sections 3.1–3.3 present the basic concepts around Autonomous operation, which allows us in Section 3.4 to discuss context diffusion.

3.1. Prerequisites for Autonomous Fog Node Operation

A Fog Node is an entity that lives in the Fog Layer and communicates with its peers in the same colony, or other layers altogether to provide services for the end-user. Some common purposes of Fog Nodes include acting as interfaces between the Cloud and the IoT, acting as computational units closer to the IoT, and finally acting as Colony Monitoring Units.

In its purest form, a Fog Node should only be able to bootstrap itself, and provide basic API Endpoints that give information about the Fog Node itself. This information can vary from Fog Colony to Fog Colony. In addition, a Fog Node should also have a purpose in the context it operates, such as the ones mentioned above. However, not all Fog Nodes have the same purpose within a Fog Colony, and not all Fog Nodes necessarily provide all of the capabilities of other Fog Nodes. Each additional capability the Fog Node should

possess, such as communicating with Wireless Sensor Networks, should be seen as extra to the basic functionality of the Fog Node.

In order to achieve the desired Context Awareness and Autonomy, the Fog Node should be able to understand its operating context, and be capable of operating with no central control, while adapting itself based on the context. The internal architecture for such a Fog Node is presented in Figure 2. It includes a number of internal functional components, or units, separated in two categories, the Core Units and the Extension Units. The Core Units provide the basic functionality of the Fog Node. Extension Units are responsible for providing new sets of capabilities to the Fog Node. However, trivially, if the functionality is not part of the main bootstrapping and basic functionality of the Fog Node, then it is considered as an additional functionality. Autonomy is to be achieved by implementing a number of Extension Units. For the Fog Node to be eligible to be considered autonomous, it needs to be able to make decisions on its own and demonstrate three characteristics, namely, Self-Configuration, Self-Adaptation, and Context Awareness. We describe Self-Configuration as the ability of the Fog Node to control and change its resources and components, such as turning off and on its CPU cores. With Self-Adaptation, we refer to the ability of the Fog Node to make self-configuration decisions based on the operating context. This adaptation is based on the needs of the moment, such as request load, energy preservation, or even needs of the Fog Colony as a whole. We define Context awareness as the ability of the Fog Node to understand the context it operates in. The context is composed of, but not limited to, information about the Fog Colony status, and information about the specific status of neighboring Fog Nodes.

Figure 2. Internal Component Architecture of the Context Aware Autonomous Fog Node.

3.2. Achieving Self-Configuration

As discussed in [19], Self-Configuration is defined as a system whose “components should either configure themselves such that they satisfy the specification or capable of reporting that they cannot”. The proposed internal architecture can be seen in Figure 2. In Figure 2, the modules relevant with Self-Configuration are the Core Components and the Internal Monitoring Module.

3.2.1. Core Components

The Control Unit is responsible for providing all the core functionalities of the Fog Node. It is responsible for initializing and bringing up the rest of the Units, and is responsible for controlling hardware aspects of the Fog Node, such as network interfaces, CPU cores, and others.

The Shared Memory Unit is the common persistence point of all units. Within the Shared Memory Unit, Key-Value pairs of information are stored, and are able to be written, read, updated, and deleted by all units of the Fog Node. The role of the Shared Memory Unit is integral to the operation of the Fog Node, as it enables the units to persist information, as well as, to exchange information in a decoupled way, and further, speeding up the Fog Node and reducing energy consumption by the unit.

The last of the Core Units is the Service Provision Unit. The Service Provision Unit is responsible for all HTTP RESTful communications of the Fog Node. It is responsible for initializing and running a low-profile web server, which provides API Endpoints the Fog Node can serve, and is also responsible for maintaining HTTP Clients that communicate with other outside entities.

3.2.2. Internal Monitoring Module

The Internal Timer and Monitoring Unit compose the Internal Monitoring Module. These Units are responsible for providing the Self-Monitoring capabilities to the Fog Node that allow it to be aware of its own operating context, or self-context. Ultimately, the module enables the Self-Configuration of the system. A Fog Node does not necessarily have to be Self-Configurable, so the units are deemed Extension Units. Although the ability to Self-Configure enables the node to change its operating parameters, this is not possible without input from the components that provide the Self-Adaptation of the node.

The Internal Timer is simply a timer that invokes the Monitoring Unit and the Decision Making Unit on set time quanta, namely the Monitoring Quantum. The Monitoring Quantum is configurable and can differ from node to node, and also be altered by policies.

The Monitoring Unit provides the ability to monitor any number of variables relevant with the operation of the Fog Node. The variables to be monitored are provided by the Self-Adaptation components, which we will discuss in the following section. The Monitoring Unit monitors the variables, called Monitored Variables, every time it is invoked by the Internal Timer. The variable values are stored in the Shared Memory Unit.

3.3. Focusing on Self-Adaptation

Self-Adaptation is defined as a characteristic of software which “evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible” [19]. The Policy Enforcement Module in Figure 2 is responsible for Self-Adaptation. This module works hand in hand with the Self-Configuration modules to enhance node autonomy. The Policy Enforcement Module is extensively discussed in [37]. The Policy Rulebook acts as a repository and holds a number of If This Then That type of instructions, that the Fog Node can enforce to its components to adapt its operation. In essence, policies are YAML-like files which describe a number of conditions which, when met, provide an action to be taken. Additionally in the policy definition, a number of Monitored Variables are defined. These variables are used by the condition statements to form simple Boolean logic, to activate the actions.

The Policy Interpreter is responsible for translating the YAML-like policy files to actual commands understood by the Fog Node. During the bootstrapping sequence, the Policy Interpreter is responsible for interpreting all available policies on the system and storing them in the Policy Rulebook. The Decision Making Unit is responsible for enforcing the actions dictated by the Self-Adaptation components. Every time the unit is invoked by the Internal Timer Unit, it checks all the relevant actions to be taken, based on the Monitored

Variables that are stored in the Shared Memory Unit. If an action should be taken based on the values of the Monitoring Variables, then it acts accordingly.

3.4. Context Awareness

We define Context, as the operational context of the node, i.e., knowledge about other Fog Nodes in the Fog Colony, as well as knowledge about their status, and provided services. In order to achieve Context Awareness, the Fog Nodes need to be able to understand and discover each others' existence, as well as to be able to communicate. Additionally, Context Aware Fog Nodes should be able to exchange knowledge amongst them, as prescribed in corresponding Policies. In this case, actions of Policies could entail communicating information to other Fog Nodes in the colony, to enhance their awareness of the operating context, and allow for enhanced cooperation between them.

In Figure 2, node-related context diffusion is served by the Context Awareness Module, composed of the Context Awareness Interface which communicates with the Context Awareness Provision Unit. The Context Awareness Provision Unit is responsible for handling all communications over the chosen context exchange protocol. This entails:

1. Propagating messages to proper recipients;
2. Receiving Context Type Messages;
3. Interpreting Context messages;
4. Keeping up to date information about context in the Shared Memory Unit;
5. Forwarding new Policies received to the Policy Interpreter.

For the Fog Nodes to be able to exchange Context Information, we need a protocol that will allow us to do so. In [20], the authors have introduced the concept of such a protocol, based on the publish/subscribe logic, aiming at exchanging information between nodes [22]. However, in the architecture presented in [20], it is proposed that a central control unit for the colony exists. This means that, since it was designed for a colony with a centralized control mechanism, the protocol has not been used to exchange information directly. Furthermore, no performance analysis was provided.

In the following chapters, we further extend the initial idea of the publish/subscribe protocol of [20], to exchange information directly between Fog Nodes. Additionally, we adjust the protocol to enable the exchange of knowledge based on Policies without a predefined message format. Finally, we explain the protocol's functionality and explore the performance it achieves in regards to the Fog Colony's operation and the load it imposes on the network.

4. ECTORAS Protocol

ECTORAS (Efficient Communication Technique Over REST Architecture Systems) Protocol is a publish/subscribe protocol based on the foundations of Data Distribution Service (DDS) [21] and RESTful communications. The main aim of the protocol is twofold:

- Distribute knowledge freely within Fog Colonies and Systems of Fog Colonies;
- Enable communications among Fog Nodes.

The protocol has two main characteristics making it suitable for exchanging knowledge directly between Fog Nodes: (a) it is based on publish/subscribe logic which allows for targeted exchange of information and (b) it allows for asynchronous exchange of knowledge between the nodes. The publish/subscribe logic of ECTORAS protocol enables nodes to create knowledge exchange topics based on their enforced Policies. This way, all nodes that enforce a specific Policy, which is enhanced by contextual information, can share this information on a separate topic. With this in mind, nodes are able to participate in all the topics they would be interested in, based on their Policies, and avoid informational overhead from contextual information that is not to their benefit.

Knowledge distribution and diffusion within Fog Systems can help accelerate certain processes that require Fog Nodes to cooperate among them. Some simple examples of such processes are task offloading, and managed resource exchange between Fog Nodes. Given

a Fog Node has prior knowledge regarding its operating context, it simplifies the process of finding suitable candidates to cooperate with. This also leads to enabling enhanced cooperative processes in the Fog layer, as is data aggregation from multiple sources to provide a single service. The aforementioned knowledge diffusion is a leading factor towards Context Aware Fog Nodes and further boosts the property of autonomy for each Fog Node of the system.

Existing protocols used in Fog and IoT environments either work in a response request manner, or if they work in a publish/subscribe manner, they do not do so without a broker. We would like to investigate the potential of publish/subscribe communications within the Fog Colony, with and without the existence of centralized entities, such as brokers. ECTORAS protocol was initially presented with a broker in mind, but is flexible enough to also work without one. To that end, we explored alternative implementations of ECTORAS protocol, both with a broker and without one, to investigate how the Fog Colonies perform.

4.1. Policies and Messages

Since the ECTORAS protocol is a publish/subscribe based protocol, Fog Nodes can create communication topics to exchange information based on their Policies. The Policies themselves dictate the information to be exchanged. That way, the creator of the Policy can define the information to be exchanged, and as such minimize and further optimize the communication overhead of the messages by tailoring the information to the needs of the Policy. Additionally, the Policies themselves define the frequency of the information to be exchanged, and whether that is to occur on specific time intervals, or when certain conditions are met. Finally, each node is able to participate in any number of topics it desires, most probably the ones relevant with the Policies it enforces, allowing for the node to skip any shared information which is irrelevant to its operating context.

As mentioned earlier, Policies are in essence YAML-like files that include a number of pieces of information, relevant to this Policy's operation and enforcement. In Figure 3, we can see a Policy example.

```
---
decision:
  actions: "OffloadContextAware.migrate()"
  conditions: "cpu_load > 0.8"
information:
  name: "Transfer Resources"
  version: "1.0"
networking:
  endpoints:
    - migrate: "MigrationEndpointHandler"
monitoring:
  variables:
    - cpu_load: "SystemInfoHelpers.getProcessCpuLoad()"
    - migration_information: "ektorasMigrationHelpers.getNodesInMigration()"
context:
  topics:
    - registration:
        group_ip: "231.3.3.7"
        group_port: "1338"
        topic_name: "resource_handover"
        message_condition: "alws_one > 0"
        message_content: "ektorasMigrationHelpers.getMessage()"
        message_handler: "ektorasMigrationHelpers.messageHandler()"
```

Figure 3. Example of deployed policy.

The Policy is broken down in a number of components, namely:

- Decision;
- Information;
- Networking;
- Monitoring;
- Context.

Each of the Policy Components defines a different functionality for the Policy. The Decision component defines what action is to be taken by the Decision Making Unit when a specific condition is met. In this case, when the Monitored Variable with name “cpu_load” is over 80%, then the Decision Making Unit is to enforce the migrate method of the bundled source code class OffloadContextAware. The Information Component provides general information about the Policy, such as its name and the current version of it. The Networking Component dictates a number of new endpoints to be provided by the Fog Node. In this case, the new RESTful endpoint migrate is to be provided, and the handler is provided by the bundled, together with the Policy, source code. The Monitoring part provides a list of variables to be monitored by the Monitoring Unit, the names of these variables, and the handlers that allow for these variables to be measured. Finally, the Context Component defines information that is going to be used by the Context Awareness Provision Unit, for context diffusion.

In Figure 4, we can see an example message exchanged over ECTORAS protocol corresponding to the context produced by the enforced policy depicted in Figure 3.

```
{
  "cpu_load": 0.56,
  "available_for_migration": true,
  "sensor_systems": [ "MicaZ", "IMA" ],
  "node": "0x27ae3c1f"
}
```

Figure 4. Example of message produced from Policy in Figure 3.

4.2. Context Exchange Mechanism

To facilitate context exchange, two different flavors of the ECTORAS protocol may be employed. To that end, two versions of the protocol were implemented: one that works in a centralized way, exploiting a broker module, and one that is brokerless, for a fully decentralized operation of the Fog Colony. We decided to support and explore both implementations, given that both have benefits and drawbacks [21,38]. Finally, we compare the two flavors’ performance when deployed in Fog Colonies of different sizes.

4.2.1. Centralized/Brokerful Implementation

In the centralized approach, the broker acts both as a repository for publish/subscribe topic information, and also as a registry where Fog Nodes can register their services. When a Fog Node is operating, it is responsible for registering its services to the broker, and is also responsible for informing the broker when it wants to participate in a publish/subscribe topic. When a Fog Node wants to send a message to a topic, then the Fog Node needs to receive the participants list from the broker, and to communicate with the relevant parties on its own, due to the DDS paradigm.

The broker is part of the Fog Colony, just as are the Fog Nodes. While it is not preferable, the role of the broker could be assumed by a Fog Node of the Fog Colony, provided that it has the computational resources needed for the task. Given the broker would probably need resources considerably higher than the average Fog Node, it is possible that not any Fog Node can take the role of the broker. To that end, it is proposed that the role of a broker is assumed by a specific entity in the system that is not a Fog Node, but an entity specifically tailored to be a broker. We assume in the rest of the paper that the broker is an entity specifically built to execute the tasks of the broker.

In Figure 5, we show an example of the communications within a colony using a broker as a central point of control. Nodes are exchanging topic information with the broker, in order to find potential participants of the topics they are interested in, and use that information to communicate in a peer to peer fashion and exchange Policy context. Specifically, Node 1 wants to send a message containing knowledge for a specific policy corresponding to a topic. To accomplish that, it communicates with the broker to receive

the latest status of the topic, and find which participants participate in it. After doing so, it finds that Node 3 and Node 4 also participate in the same topic and sends the relevant message to them. Node 2 participates in the colony but for its operation it does not need to act in a context aware manner. To that end, Node 2 does not participate in active knowledge sharing, and therefore, it does not communicate with any other node in the figure.

Figure 5. Example communications using a broker and ECTORAS protocol.

One of the benefits of having a centralized approach is that the system complexity of the Fog Colony is low. All Fog Nodes reach the broker to find the participants per topic, and send their messages to them directly. Additionally, it keeps the operating costs of the Fog Nodes low, as they do not have to keep a lot of information about the topics in their end, as the broker takes care of this. Moreover, they do not have to keep track of updates and changes in the topic. Finally, service discovery is rather straightforward, with internal to the Fog Layer entities, but also external ones, only needing to refer to the broker to find any service required.

The drawbacks of having a centralized unit are also important to mention. Initially, a centralized architecture brings a single point of failure, where if the broker was to fail, the whole Fog Colony would instantly lose the ability to cooperate. Additionally, given the fact that Fog Colonies can be extensive in size, the broker would be under heavy load and require a significant amount of resources to keep up with the load. Given the load the centralized unit would have to handle, it also brings the question as to the scalability of a centralized system. In addition, by relying on a broker to enable context diffusion, time critical context information might be slower to reach their intended targets, and therefore, significantly increasing lookup times for cooperative processes. Finally, by not having contextual information always readily available when needed, and having to go through big lookup times, including communicating with the central authority, to find suitable Fog Nodes to cooperate with, time-critical operations are at risk. To sum up, these limitations can be formulated as:

1. Single point of Failure for the whole colony;
2. Heavy resource cost for the centralized unit;
3. Scalability concerns;
4. High lookup times for cooperative processes;

5. Time critical operations at risk.

4.2.2. Decentralized/Brokerless Implementation

In the decentralized approach, no broker is present. This comes with a number of challenges to be tackled. With no broker present, it is impossible for any entity to know the contextual status of the Fog Colony. Additionally, service discovery becomes impossible, and it goes without saying that the Fog Nodes would not be able to cooperate or know any operational context. To fill in the gap created by the removal of the broker, we propose two actions. First, the Fog Nodes themselves should internalize part of the role of the broker, to the extent that they keep their own copy of the operating context, service information, and topic information. Second, a new communication pattern needs to be introduced.

Given all communications in the Fog Colony architecture presented are based on the Internet protocol suite, we could exploit this to our benefit. The Internet protocol suite, allows for a number of IP addresses to be used as broadcast addresses [39]. A broadcast message is a message that is sent by a member of the network, and is received by every member on the network, given they are listening for messages on the specific address. We propose that a specific broadcast IP address could be used for the whole Fog Colony to be able to exchange information which would otherwise be handled by the broker. Fog Nodes would emit a heartbeat type of message on this IP address at specific time intervals, and other Fog Nodes are responsible for listening for those messages and handling them accordingly. We call this communication pattern the Colony Hall, where all Fog Nodes congregate to align.

The messages exchanged at the Colony Hall should be short and concise, to avoid stuffing the communication channel. The messages should include basic information about the Fog Node, such as a Unique Identifier of the Fog Node, the IP address the Fog Node is reachable at, the services it provides, and the time the message was emitted. Other Fog Nodes should be able, based on this information, to create a record in their Shared Memory unit that represents the Fog Node operating. Similarly, if a Fog Node fails to emit a message for a certain time period, or a Fog Node fails to receive messages from a specific other Fog Node within a certain time period, that Fog Node should be considered out of the current Fog Nodes operating context.

In Figure 6, we can see an example of communications using the Colony Hall paradigm to exchange messages. In the figure, we can see Node 1 sending a heartbeat to the Colony Hall, while Nodes 3 and 4 are listening to the Colony Hall for new messages. Similarly to Figure 5, when Node 1 needs to send to a specific topic a message containing contextual information, it needs to find who participates in this topic. By internalizing the broker functionalities to each node, the node already has this information, and continues to directly communicate with Node 3 and Node 4, to exchange Policy Knowledge. Again, like in Figure 6, Node 2 does not need to exchange Policy Knowledge to fulfill its operations, and thus does not participate in the Colony Hall.

The benefits of following a decentralized approach have to do a lot with the flexibility and scalability of the system. Initially, by not being bound by a centralized entity, Fog Nodes can truly operate in an autonomous manner, fully exploiting their context information. Next, Fog Nodes are able to join, exit, and jump between colonies much easier, only needing to adjust their operating context, without having to inform a centralized entity. This also affects the Fog Colony as a whole, since it improves the flexibility of the colony, allowing for Fog Nodes to roam in the Fog Layer, and can scale up virtually infinitely, since there is no central authority to control the colony. Lower energy consumption is also expected, since the Fog Nodes will be able to more easily identify Fog Nodes they can cooperate with, and communications will be more concise. Finally, one of the most important benefits is the removal of the single point of failure that the broker poses.

Figure 6. Example communications using the Colony Hall and ECTORAS protocol.

Nonetheless, there are also drawbacks in using a decentralized architecture. Initially, the Fog Colony system is much more complex, with Fog Nodes needing to keep themselves up to date with all changes in the Fog Colony, and managing their operating context completely. An additional communication mechanism needs to be present, the colony hall, which is used by the Fog Nodes to emit their heartbeat to the rest of the colony. This heartbeat is used by the rest of the Fog Nodes to maintain a complete picture of the Fog Colony, something that in the brokerful implementation the broker is tasked with doing. Moreover, in small Fog Colony sizes, we expect the energy consumption to be significantly higher than that of the centralized architecture, since the major overhead of the centralized architecture comes during the time when a Fog Node is executing a cooperative task and is looking for a suitable collaborator. If the size of the colony is small, there are not many nodes that can be potential candidates, and therefore, the overhead is limited. Finally, we expect a similar result for the network bandwidth utilization for the same reasons, and given that, the decentralized approach has a constant heartbeat from the Fog Nodes that utilizes the network.

5. Performance Testing

Fog Colonies operate in high traffic environments, providing a variety of services with different quality of service requirements. Thus, one of the main considerations for the performance of the Fog Nodes is the ability to serve requests rapidly. If they are not able to serve a request fast enough, they should explore ways to mitigate this. One way to do so, is to offload the task to another Fog Node. The speed with which task offloading can be completed is important for the quality of service preservation of the whole Colony. Such a task is vastly enhanced by contextual information, given that by exploiting contextual information, fog nodes can more easily and quickly determine which potential candidates are more viable to offload the task to. Tasks that require fog nodes to cooperate will be referred to as Cooperative Tasks. As discussed, the aim of this work is to explore the diffusion of context in Fog Colonies using the publish/subscribe communication paradigm, with both a broker present, and without. To that end, two different flavors of the ECTORAS protocol in different Fog Colony scenarios were implemented. To test the two different implementations of ECTORAS protocol with regards to the quality of service concerns

mentioned above, the performance of the Fog Colonies is inspected and compared in regard to the following aspects:

1. The difference in completion time for Cooperative Tasks within the Fog Colony;
2. The difference in overall Fog Node energy consumption;
3. The difference in the total number of network operations per Fog Node.

In this chapter, we present our findings of the performance testing, while in the next section, we shall discuss them.

For each of the metrics, we conducted a series of experiments with the set up as follows. We developed a prototype implementation of the envisioned architecture presented in Figure 2 using Java 15. Two flavors of the Context Awareness Provision Unit were implemented, where one worked with the existence of a broker, and the other one worked in a brokerless manner. The middleware was deployed on Raspberry Pi 3 revision A units (one node per Raspberry Pi). The units were linked to each other via local Ethernet network, using a number of 10/100/1000 Mbps switches. In the end of the setup, we were able to construct two Fog Colony systems, one using the brokerful version of ECTORAS, and one using the brokerless version of ECTORAS. Although the nodes were deployed, no sensors were connected to them, to simplify the effort of setting up the system. To create load on Fog colonies, we implemented a load generator that requested services from the nodes based on certain user-defined profiles.

We defined 4 sizes of colonies. A tiny colony consisted of 3 Fog Nodes, a small-sized colony consisted of 15 nodes, a medium colony consisted of 30 nodes, and a large colony consisted of 50 nodes. For all the experiments, we set a time limit for observation of 60 min and for each of the metrics we wanted to examine, we ran tests on all sizes of colonies, and compared the average values for each outcome. The Policy seen in Figure 3 was used to create and manage topics.

5.1. Average Task Completion Time for Cooperative Tasks

One of the main advantages of the brokerless flavor is that Fog Nodes internalize their operating context. That way, when needed to execute cooperative processes, the Fog Nodes are able to select a potential cooperator in a smarter and faster fashion, without needing to ask every fog node available in their colony.

In Figure 7, the horizontal axis represents the colony size, while the vertical represents the completion time in ms for cooperative tasks. The figure is a box plot of the data where boxes represent the First and Third quartile of data, while the thick black lines signify the median of the data. The whiskers on the top and bottom of the boxes show the minimum and maximum values of the data.

For the tiny colony size, the brokerful system layout had an average completion time of cooperative tasks of 62 ms, while the brokerless system had an average of 37 ms. For the small colony size, the average for the brokerful system layout was 1223 ms, while for the brokerless system was 273 ms. For the medium-sized colony, the average for the brokerful layout was 1630 ms, while for the brokerless system it was 350 ms. Finally, for the large colony, the brokerful layout had an average of 2383 ms, while the brokerless layout had an average of 398 ms.

5.2. Average Lifetime per Node

One major concern for the brokerless system is the energy consumption of the Fog Nodes. Given Fog Nodes can be deployed virtually anywhere, it is common that they run on battery power. To that extent, the energy consumption of the Fog Nodes is something that is important to be kept in mind. In Figure 8, the horizontal axis represents the colony size, while the vertical represents the time in minutes a Fog Node was able to sustain itself running on battery power. For the tiny colony size, as regards the brokerful system layout, the average operation time was 819 min, while for the brokerless system layout the average was 746 min. For the small colony size, the brokerful system layout achieved an average of 519 min while the brokerless system layout achieved an average operation

time of 624 min per node. For the medium colony, the brokerful system layout achieved an average running time of 116 min, while the brokerless system layout achieved an average of 442 min. Finally, for the large colony, the brokerful system achieved an average of 93 min, while the brokerless system achieved an average of 388 min.

Completion time for Cooperative Tasks within the Fog Colony for the Brokerless System

Figure 7. Box chart showing the comparison on the completion time for cooperative task execution between the two Fog Colony Architectures.

Lifetime of Node running on Battery Power for the Brokerless System

Figure 8. Box chart showing the comparison on the lifetime per node between the two Fog Colony Architectures.

5.3. Average Number of Network Operations

While the available network bandwidth in Fog Systems is not a major concern, we wanted to be able to inspect the potential additional load the brokerless layout could bring. To that extent, we set off to calculate the average number of network operations exhibited by each Fog Node per minute of operation. We count as a network operation every incoming network connection and every outgoing network connection, but we do not take into consideration the network traffic that was generated by the load generator, as we aim to test the internal to the Fog Colony network operations.

In Figure 9, the horizontal axis represents the colony size, while the vertical represents the number of network operations per minute, per node. For the tiny colony size, for the brokerful system layout, the average number of network operations per minute was 108, while for the brokerless system layout, the average was 1668 operations. For the small colony size, the brokerful system layout achieved an average of 2497 operations, while the brokerless system layout achieved an average of 2979 operations. For the medium colony, the brokerful system layout achieved an average of 5135 operations, while the brokerless system layout achieved an average of 4169 operations. Finally, for the large colony, the brokerful system achieved an average of 6110 operations, while the brokerless system achieved an average of 4569 operations.

Figure 9. Box chart showing the comparison on the network operations per minute per node between the two Fog Colony Architectures.

6. Discussion

The findings of the performance tests for the two flavors of the ECTORAS protocol, relating to the two different techniques of diffusing context, revealed the following:

Exploring the Average completion time for cooperative tasks, we can notice the following. In general, the brokerless system performed better by a significant factor. Although for the tiny colony, the completion times were very close, as the colonies grew in size, the difference was apparent. This seems to relate to the existence of the broker. The more the colony grew in size, the more nodes needed to communicate with the broker. This had an effect on the response times of the broker, slowing down the colony operations in general.

Looking at the Average lifetime of the Fog Nodes, we can interestingly see that for the tiny colony size, the brokerful system architecture performed better. This is due to the fact that less power would have to be consumed to maintain the records in the Registry Unit (write once and forget), while for the brokerful system, regardless of the size of the colony, the nodes had to send the broadcast messages. Nonetheless, as the colonies grew in size, the gains from the brokerless system became apparent. Specifically, for the large colony, the gains of the brokerless system are nearly quadruple. This seems to be related directly with the process of fetching topic information from the broker for every update the nodes wanted to send to any topic.

Finally, regarding the number of network operations, we see that the brokerful system has a significantly lower number of network operations for the tiny colonies. The brokerful system continues to achieve a lower number of network operations in comparison to the brokerless system for the small colony, but the two are much closer. For the medium- and large-sized colonies, the brokerless system performs better. This is an interesting result, but expected. Given that in the brokerful system, each time someone needs to send a message, the nodes need to communicate with the broker to find the current participants of each topic. As a result, the more the size grew, the more communications started appearing towards the broker. Additionally, and interestingly, when the broker started experiencing slow downs due to the size of the colony, at times, the nodes would have to resend requests to the broker, as the requests would time out due to the slowdown of the broker. On the other hand, due to the way the Colony Hall operates, in the brokerless system, all nodes would either way update all participating nodes of their current status, regardless of the colony size. However, as the colonies grew, the number of updates per node in the brokerless system did not grow significantly.

Based on these findings, we see that for really small colonies, the centralized architecture performs better in all cases. However, the centralized system still has the risk of the single point of failure. As the Fog Colony grows in size, the brokerless system seems to be preferable, and it gains in performance, yielding better results for energy consumption and bandwidth usage, and more significantly improving cooperation times between the Fog Nodes.

The most important gain, however, is the fact that Fog Nodes running on the brokerless system have now achieved complete autonomy, and can operate with no central control. They are able to choose for themselves how to handle anything their Policies define, and are self-aware of their operational context. Additionally, service discovery is very simple, with one needing only to subscribe to the Colony Hall to receive updates for all provided services in the colony.

7. Conclusions—Future Work

Promoting the autonomous operation of Fog Nodes, though it may boost the effectiveness and extendability of fog colonies, is not a trivial task. Besides running control middleware in autonomous nodes to achieve Self-Control and Self-Adaptation, awareness of the colony context they share should be achieved. To this end, we described and implemented two flavors of the ECTORAS protocol, to facilitate knowledge exchange between Fog Nodes, one using a broker, and one without a broker. Our main goal was to explore different techniques for context diffusion within Fog Colonies, with our ultimate goal being the autonomous operation of Fog Nodes; thus, allowing for more flexibility and scalability for the Fog Colonies.

Performance tests, conducted on the prototype implementation of the Autonomous Fog Node Control middleware utilizing both versions of ECTORAS, showed that, while for small colonies utilization of the brokerful version is preferable, as the colony's size grew, the gains from employing a brokerless approach became apparent. The gains at scale were due to the fact that, eventually, the broker became a congestion point of the colony, slowing down the colony operations as a whole.

As part of future work, we plan to explore the following issues: (a) further study the limitations of decentralized Fog Colonies as their size expands, and members frequently move from one colony to another. (b) Explore the ability of Fog Nodes to exchange Policies within a Fog Colony. This should come hand in hand with Policy evaluation mechanisms, as well as, communication mechanisms for exchanging the Policies. (c) Apply the Policy concept to control decentralized data in the Fog Layer, as in decentralized databases. Given that the Fog Layer acts as a data pre-filter for the Cloud, it would make sense to exploit the nodes to deploy distributed processing of large datasets. Policies could be used to orchestrate the control of the data distribution.

Author Contributions: Conceptualization was done by V.N., M.N. and M.V.; Investigation was carried out by M.V. and V.N.; Project administration was performed by M.N., D.A. and V.N.; Software was developed and tested by V.N.; Supervision by D.A. and M.N.; Validation by M.N.; Visualization executed by M.V. and V.N.; Writing—original draft by V.N. and M.V.; Writing—review & editing by V.N., M.N. and M.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; pp. 13–16.
2. Cha, H.J.; Yang, H.K.; Song, Y.J. A Study on the Design of Fog Computing Architecture Using Sensor Networks. *Sensors* **2018**, *18*, 3633. [CrossRef]
3. Hussein, M.K.; Mousa, M.H. Efficient task offloading for IoT-based applications in fog computing using ant colony optimization. *IEEE Access* **2020**, *8*, 37191–37201. [CrossRef]
4. OpenFog Consortium. OpenFog Reference Architecture, 2017. Available online: https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf (accessed on 4 January 2022).
5. Richardson, L.; Ruby, S. *RESTful Web Services*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2008.
6. Shelby, Z.; Hartke, K.; Bormann, C. The Constrained Application Protocol (CoAP). 2014. Available online: https://iottestware.readthedocs.io/en/master/coap_rfc.html (accessed on 4 January 2022).
7. Bormann, C.; Castellani, A.P.; Shelby, Z. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Comput.* **2012**, *16*, 62–67. [CrossRef]
8. Chegini, H.; Naha, R.K.; Mahanti, A.; Thulasiraman, P. Process Automation in an IoT–Fog–Cloud Ecosystem: A Survey and Taxonomy. *IoT* **2021**, *2*, 92–118. [CrossRef]
9. Ayoubi, M.; Ramezanpour, M.; Khorsand, R. An autonomous IoT service placement methodology in fog computing. *Softw. Pract. Exp.* **2021**, *51*, 1097–1120. [CrossRef]
10. Tran, M.Q.; Nguyen, D.T.; Le, V.A.; Nguyen, D.H.; Pham, T.V. Task placement on fog computing made efficient for iot application provision. *Wirel. Commun. Mob. Comput.* **2019**, *2019*, 6215454. [CrossRef]
11. Nguyen, Q.H.; Pham, T.A.T. Studying and developing a resource allocation algorithm in fog computing. In Proceedings of the 2018 International Conference on Advanced Computing and Applications (ACOMP), Ho Chi Minh City, Vietnam, 27–29 November 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 76–82.
12. Al-Khafajiy, M.; Baker, T.; Al-Libawy, H.; Maamar, Z.; Aloqaily, M.; Jararweh, Y. Improving fog computing performance via fog-2-fog collaboration. *Future Gener. Comput. Syst.* **2019**, *100*, 266–280. [CrossRef]
13. Batool, A.; Loke, S.W.; Fernando, N.; Kua, J. Towards a Policy Management Framework for Managing Interaction Behaviors in IoT Collectives. *IoT* **2021**, *2*, 633–655. [CrossRef]
14. Yousefpour, A.; Ishigaki, G.; Jue, J.P. Fog computing: Towards minimizing delay in the internet of things. In Proceedings of the 2017 IEEE International Conference on Edge Computing (EDGE), Honolulu, HI, USA, 25–30 June 2017; IEEE: Piscataway, NJ, USA; pp. 17–24.
15. Giordano, A.; Spezzano, G.; Vinci, A. Smart agents and fog computing for smart city applications. In *International Conference on Smart Cities*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 137–146.
16. Shaik, S.; Baskiyar, S. Hierarchical and Autonomous Fog Architecture. In Proceedings of the 47th International Conference on Parallel Processing Companion, Eugene, OR, USA, 13–16 August 2018; ACM: New York City, NY, USA, 2018; p. 18.
17. Baek, J.y.; Kaddoum, G.; Garg, S.; Kaur, K.; Gravel, V. Managing fog networks using reinforcement learning based load balancing algorithm. In Proceedings of the 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, 15–18 April 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–7.

18. Baranwal, G.; Vidyarthi, D.P. FONS: A fog orchestrator node selection model to improve application placement in fog computing. *J. Supercomput.* **2021**, *77*, 10562–10589. [\[CrossRef\]](#)
19. Nikolopoulos, V.; Dimopoulos, A.C.; Nikolaidou, M.; Dimitrakopoulos, G.; Anagnostopoulos, D. A system of systems architecture for the internet of things exploiting autonomous components. *Int. J. Syst. Syst. Eng.* **2019**, *9*, 167–199. [\[CrossRef\]](#)
20. Nikolopoulos, V.; Voreakou, M.; Nikolaidou, M.; Anagnostopoulos, D. Enhancing Context-Awareness in Autonomous Fog Nodes for IoT Systems. In Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE), Milan, Italy, 8–13 July 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 113–115.
21. Pardo-Castellote, G.; Chairman, D. Omg Data Distribution Service: Real-Time Publish/Subscribe Becomes a Standard. Available online: https://www.rti.com/hubfs/docs/reprint_rti.pdf (accessed on 4 January 2022).
22. Kaushik, S.; Poonia, R.C.; Khatri, S.K. Comparative study of various protocols of DDS. *J. Stat. Manag. Syst.* **2017**, *20*, 647–658. [\[CrossRef\]](#)
23. Pardo-Castellote, G. Omg data-distribution service: Architectural overview. In Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, Providence, RI, USA, 19–22 May 2003; IEEE: Piscataway, NJ, USA, 2003; pp. 200–206.
24. Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Jcatascopia: Monitoring elastically adaptive applications in the cloud. In Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA, 26–29 May 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 226–235.
25. Skarlat, O.; Nardelli, M.; Schulte, S.; Dustdar, S. Towards qos-aware fog service placement. In Proceedings of the 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), Madrid, Spain, 14–15 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 89–96.
26. Zhang, G.; Shen, F.; Yang, Y.; Qian, H.; Yao, W. Fair task offloading among fog nodes in fog computing networks. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 1–6.
27. Skarlat, O.; Karagiannis, V.; Rausch, T.; Bachmann, K.; Schulte, S. A framework for optimization, service placement, and runtime operation in the fog. In Proceedings of the 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), Zurich, Switzerland, 17–20 December 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 164–173.
28. Yousefpour, A.; Ishigaki, G.; Gour, R.; Jue, J.P. On reducing IoT service delay via fog offloading. *IEEE Internet Things J.* **2018**, *5*, 998–1010. [\[CrossRef\]](#)
29. Nikolopoulos, V.; Dimopoulos, A.C.; Nikolaidou, M.; Dimitrakopoulos, G.; Anagnostopoulos, D. The role of autonomous aggregators in IoT multi-core systems. In Proceedings of the Seventh International Conference on the Internet of Things, Linz, Austria, 22–25 October 2017; pp. 1–8.
30. Balevi, E.; Gitlin, R.D. Optimizing the number of fog nodes for cloud-fog-thing networks. *IEEE Access* **2018**, *6*, 11173–11183. [\[CrossRef\]](#)
31. Skarlat, O.; Schulte, S.; Borkowski, M.; Leitner, P. Resource provisioning for IoT services in the fog. In Proceedings of the 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, China, 4–6 November 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 32–39.
32. Karagiannis, V.; Schulte, S.; Leitao, J.; Preguiça, N. Enabling Fog Computing using Self-Organizing Compute Nodes. In Proceedings of the 2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC), Larnaca, Cyprus, 14–17 May 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1–10.
33. Dey, A.K. Understanding and using context. *Pers. Ubiquitous Comput.* **2001**, *5*, 4–7. [\[CrossRef\]](#)
34. De Prado, A.G.; Ortiz, G.; Boubeta-Puig, J. CARED-SOA: A context-aware event-driven service-oriented Architecture. *IEEE Access* **2017**, *5*, 4646–4663. [\[CrossRef\]](#)
35. Ortiz, G.; Zouai, M.; Kazar, O.; Garcia-de Prado, A.; Boubeta-Puig, J. Atmosphere: Context and situational-aware collaborative IoT architecture for edge-fog-cloud computing. *Comput. Stand. Interfaces* **2022**, *79*, 103550. [\[CrossRef\]](#)
36. Naik, N. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In Proceedings of the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria, 11–13 October 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1–7.
37. Nikolopoulos, V.; Voreakou, M.; Nikolaidou, M.; Anagnostopoulos, D. Fog Node Self-Control Middleware: Enhancing Context Awareness towards Autonomic Decision Making in Fog Colonies. *Elsevier Internet Things* **2021**. *Under Submission*.
38. Abegg, J.P.; Bramas, Q.; Brugiére, T.; Noel, T. SUPRA, a distributed publish/subscribe protocol with blockchain as a conflict resolver. In Proceedings of the 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS), Paris, France, 27–30 September 2021; pp. 41–42. [\[CrossRef\]](#)
39. Forouzan, B.A. TCP/IP Protocol Suite. Available online: <https://www.mheducation.com/highered/product/tcp-ip-protocol-suite-forouzan/M9780073376042.toc.html> (accessed on 4 January 2022).