Accepted at: 4th IEEE International Conference on Fog and Edge Computing (ICFEC 2020), pp. 1–10

Comparison of Alternative Architectures in Fog Computing

Vasileios Karagiannis, Stefan Schulte
Distributed Systems Group, TU Wien, Austria
{v.karagiannis, s.schulte}@dsg.tuwien.ac.at

***Abstract*—Since the proliferation of fog computing, various distributed architectures have been proposed to extend the cloud to the edge of the network. However, so far there exists no study that compares different fog computing architectures, and produces quantitative results in order to examine the efficiency of each architecture for different use cases. Such a study could provide guidelines for selecting an appropriate distributed architecture for fog computing while taking into account the requirements of the final applications.**

To bridge this gap in the literature, we create a unified system model which is able to represent the basic architectures commonly used for fog computing, i.e., hierarchical and flat. Furthermore, we design algorithms that can be used for creating fog computing systems that follow these architectures, and we perform various experiments that focus on communication latency and bandwidth utilization. Notably, our results show that for applications that do not have a dependency on the cloud, i.e., no resource-demanding tasks are involved, the hierarchical architecture reduces the communication latency by 13% compared to the flat. However, for applications that also include resource-demanding tasks, the flat architecture reduces the communication latency by 16% compared to the hierarchical.

***Index Terms*—Fog computing, edge computing, distributed architectures, hierarchical architecture, flat architecture**

I. Introduction

In the Internet of Things (IoT), appliances may communicate with each other in order to enable smart applications [1]. For example, smart traffic lights can interact with the vehicles that roam the streets in order to enable traffic management according to the actual traffic, and not based on fixed time intervals [2]. Garbage collection trucks can interact with the dumpsters in order to configure their schedule to avoid the accumulation of garbage, rather than operating on static routes [3]. Similarly, various applications can be complemented by sensor observations, such as environmental monitoring, healthcare, and smart grids, among others [4]–[7]. However, the computational resources of the appliances alone, may not be sufficient to facilitate such smart applications. For this reason, the appliances usually communicate with each other through the cloud [8].

With the help of the cloud, the IoT became very successful which led to more and more smart applications utilizing more and more appliances (also referred to as smart things, or IoT devices) [9]. Since the cloud represents computational resources in data centers, it provides virtually unlimited compute capacity [10]. Nevertheless, high communication latency between the IoT devices and the cloud, along with bandwidth limitations and privacy concerns, led to the rise of fog computing [11].

Fog computing emerged to deal with these issues by exploiting various compute nodes between the cloud and the smart things, as shown in Fig. 1. These compute nodes may be, e.g., base stations and access points as well as cloudlets and fog nodes which reside at the edge of the network [12], [13]. By leveraging on such nodes, fog computing processes the IoT data close to the data source, and only incorporates cloud-based computational resources if necessary [14]. Furthermore, since these compute nodes reside in closer proximity to the smart things than the cloud, the processing takes place with lower communication latency, and with higher bandwidth capacity [15].

Many fog computing architectures have been proposed so far, aiming at leveraging the edge of the network in order to distribute the processing of the IoT data, and to lower the communication latency [16]. Most of these architectures describe fog computing systems with compute nodes which are organized hierarchically in layers [17], [18]. According to the hierarchical architecture, the compute nodes of the cloud reside at the top of the hierarchy, the compute nodes at the edge of the network in the middle, and the IoT devices at the very bottom [19]. The IoT devices usually send the data to the compute nodes at the edge. However, connections to other nodes (e.g., to the cloud) are also possible.

Alternatives to the hierarchical architecture have also been proposed in fog computing. The alternatives consist of flat architectures whereby the compute nodes form connections to each other, and communicate without the use of layers [20]. According to the flat architecture, each node forms connections to a limited number of other nodes (i.e., neighbors), and uses the neighbors in order to send data to nodes that reside farther away [21].

Even though the aforementioned alternative architectures have been proposed for fog computing, it is still unclear under which conditions each one should be preferred. For this reason, in this paper we create a unified system model which is able to represent the basic fog computing architectures, i.e., hierarchical and flat. Based on this model, we create fog computing systems that follow these architectures, and we perform various experiments regarding communication

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA—Fog Computing for Robotics and Industrial Automation.

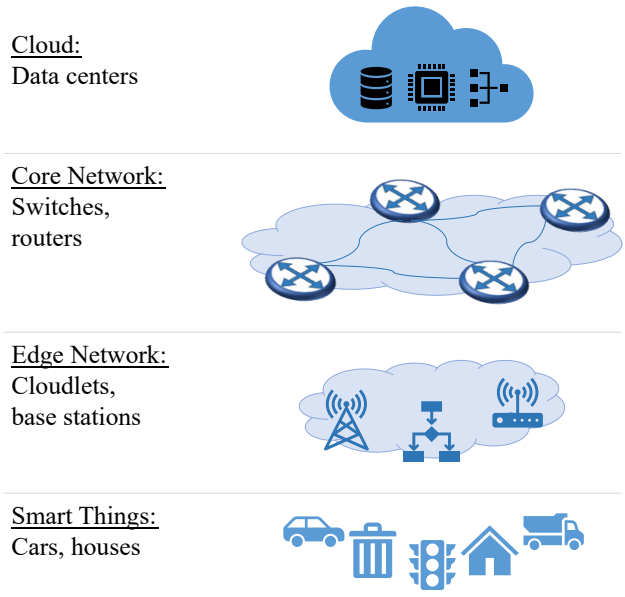


Fig. 1: Compute nodes in fog computing.

latency and bandwidth utilization. Along with the system model and the quantitative results which can be used for comparing the different architectures, our contributions also include a discussion of these results. This discussion provides guidelines for selecting an appropriate architecture based on the requirements of the final applications in mind.

The rest of this paper is organized as follows: Section II contains the utilized system model, and an analysis of the basic fog computing architectures. Consequently, Section III presents an implementation of these architectures, and provides experimental results which aim at highlighting the main differences regarding communication latency, and bandwidth utilization. Finally, Section IV presents a discussion of related work, and Section V concludes this paper.

II. Fog Computing Architectures

This section aims at presenting hierarchical and flat architectures for fog computing. To this end, Section II-A provides a unified system model, and basic definitions which are used hereinafter. Afterwards, Sections II-B and II-C present hierarchical and flat architectures, respectively.

A. System Model

In this section, we design a system model which is based on basic principles from fog computing system models from the literature (cf. Section IV). However, compared to these models, our system model is designed to be able to represent various fog computing architectures. The basic building block of our model is a compute node. A compute node may reside in different parts of the network, i.e., at the edge of the network (e.g., in a cloudlet or a fog node), at the access network (e.g., in an access point or a base station), the core network (e.g., in a router or a gateway), or at the cloud (e.g., in a data center).

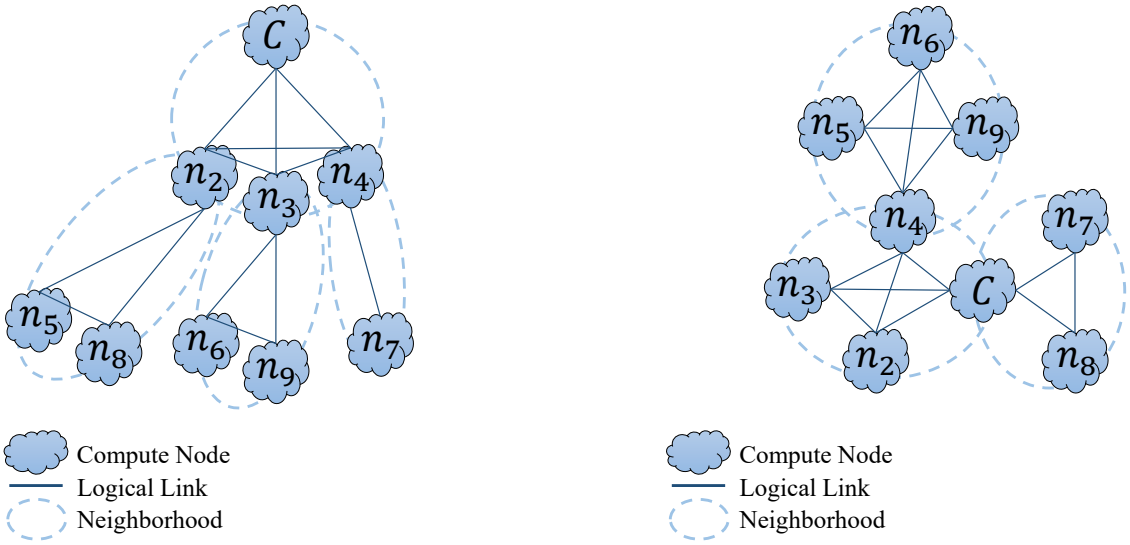
The compute nodes of the system are denoted as $n_1, n_2, \ldots, n_N$. The cloud is considered to be a compute node which is different from the others, in that it is responsible for global coordination [9]. For this reason, the cloud is used as an entry point for other nodes to join. Since the cloud is used as an entry point, we assume that the cloud is the first compute node n_1. To distinguish it from the other compute nodes, the cloud node is also denoted as C.

As fog computing architecture, we define the manner whereby the compute nodes communicate with each other in order to process the data from the IoT devices [22]. In the hierarchical architecture, the participating compute nodes are organized hierarchically in layers, and each node communicates with the nodes of the adjacent layers. In the flat architecture, there are no layers. Instead, each node communicates with other nodes in a topology that resembles a mesh network [21]. Independently of the utilized architecture, all the IoT devices together, along with all the participating compute nodes, are referred to as a fog computing system.

In order to be able to represent both hierarchical and flat architectures with the same model, we use the notion of a neighborhood H. A neighborhood includes different compute nodes which are connected in a complete graph, i.e., each pair of compute nodes in a neighborhood is connected by an arc. These arcs represent the logical links among the nodes, and can be either weighted or unweighted. When the arcs are weighted, the values of the weights represent a proximity measure (e.g. hop count or round-trip time). When the arcs are not weighted, the compute nodes are agnostic of the exact proximity to other nodes. The compute nodes that belong to the same neighborhood are also referred to as neighbors. Neighborhoods can have up to m neighbors. Thus, m is a system parameter which can be used for changing the number of neighbors that each node can have.

An example of a hierarchical architecture is shown in Fig. 2a which shows nine compute nodes organized hierarchically in three layers with a maximum neighborhood size m that equals four. The compute node at the top of the hierarchy always belongs to only one neighborhood (e.g., C in Fig. 2a). By considering the hierarchy as a tree, the node at the top (i.e., the root of the tree) can also be regarded a parent, while the nodes below are children, e.g. in Fig. 2a, C is the parent, and n_2, n_3, and n_4 are children. Each one of the nodes in the middle layers belongs to two neighborhoods. One neighborhood includes the parent and siblings (i.e., same parent), e.g., n_2 belongs to a neighborhood that also includes C, n_3 and n_4. The other neighborhood includes children, e.g., n_2 belongs to a neighborhood that also contains n_5 and n_8. Finally, the nodes at the edge of the network which reside at the leaves of the hierarchy, belong to only one neighborhood in which they are children. For example, as shown in Fig. 2a, each one of the nodes n_5, n_8, n_6, n_9, and n_7 belong to only one neighborhood.

In the flat architecture, the number of neighborhoods that a node can belong to, is not limited [21]. Furthermore, in this architectural style, there are no parents, siblings, and children because the architecture no longer resembles a tree. Instead, every node that belongs to more than one neighborhoods, is



(a) A hierarchical fog computing system.

(b) A flat fog computing system.

Fig. 2: Fog computing systems according to different architectures (with neighborhood size that equals four).

also referred to as a liaison. For instance, in Fig 2b which shows nine compute nodes which are organized according to the flat architecture, C and n_4 are liaisons. Liaisons are necessary because they are used for propagating data from one neighborhood to the other(s). Thus, every neighborhood needs to have at least one liaison so that the IoT data can be propagated to the other neighborhoods, and consequently to the rest of the fog computing system. Notably, in contrast to the hierarchical architecture which expands downwards, the flat architecture can expand towards any direction.

Independently of the utilized architecture, the compute nodes of a fog computing system need to host various applications in order to process the IoT data. To achieve this, we assume an application model which is based on a widely used model for fog computing from the literature [23]. Every application consists of tasks. Each task can have different latency requirements based on the functionality of the corresponding application. For this reason, a task can be either delay-sensitive (e.g., for augmented reality applications) or delay-tolerant (e.g., for analytics). The delay-sensitive tasks are usually deployed on the compute nodes close to the IoT devices so that the IoT data can be processed with low communication latency. The delay-tolerant tasks, on the other hand, can be deployed on any compute node including the cloud.

In this model, the flow of the data starts from the IoT devices which integrate sensors (that generate IoT data), and send the IoT data to the compute nodes in proximity for processing. If the processing cannot take place because these nodes are busy (i.e., the nodes do not have enough available computational resources), the nodes forward the IoT data to their neighbors. If the neighbors cannot process the data either, then the data is propagated to another neighborhood. This is done as follows: In the hierarchical architecture, if the data cannot be processed within a neighborhood, the parent node propagates the data to the other neighborhood that it belongs to. This results in propagating the data upwards the hierarchy. In the flat architecture, a liaison node is responsible for propagating the data to the other neighborhood(s) that it belongs to. This process continues for as long as the nodes are busy, and until the data reaches the cloud which is always able to process the IoT data due to having virtually unlimited computational resources [10].

There are various placement algorithms in the literature for placing the tasks of an application on compute nodes that span from the cloud to the edge of the network (e.g., [24], [25]). Notably, such approaches usually require the graph of the compute nodes (i.e., the neighborhoods) as input. Thus, our work can be considered complementary to application placement approaches.

B. The Hierarchical Architecture

According to the hierarchical architecture, the participating compute nodes are organized in two, three, or more layers. Commonly, the nodes that reside at the edge of the network close to the IoT devices, are placed at the lowest layer, and are only able to provide limited computational resources [26]. For this reason, the IoT data can be processed there with low communication latency although, resource-demanding processing might not be possible on these nodes. Nevertheless, while the IoT data is propagated upwards the hierarchy (as discussed in Section II-A), nodes with more computational resources are found, but the communication latency increases as well. Due to having the IoT devices send the data to the compute nodes of the closest proximity, which integrate limited computational resources, this architecture becomes appropriate for applications with tasks that are not resource-demanding, and require low communication latency. Nevertheless, there have

Algorithm 1: HNP

```
1  joinHierarchical(computeNode C){
2  joinResponse = joinRequest(C)
3  if joinResponse == ∅ then
4  |   H_new.add(this, C)
5  |   H_new.updateNeighbors()
6  else if joinResponse == H then
7  |   H.add(this)
8  |   H.updateNeighbors()
9  else if joinResponse == n then
10 |   joinHierarchical(n)
11 end
12 }
```

also been approaches that try to improve the management of the available computational resources by placing the tasks on compute nodes at the edge and in the cloud selectively [27].

To represent the hierarchical architecture in fog computing, we design HNP (Hierarchical architecture No Proximity) which is described in Algorithm 1, and can be used for organizing the participating compute nodes in layers. This algorithm can represent hierarchical architectures of any number of layers, because the number of layers of the resulting fog computing system, depends on the number of the participating compute nodes. Notably, HNP is proximity agnostic. This means that the neighborhoods are represented by unweighted graphs, and that the nodes know their neighbors, but not their exact proximity [28].

In the following, we describe how HNP operates. To make HNP more comprehensible, Algorithm 1 explains the steps of the algorithm from the perspective of a new compute node that requests to join the fog computing system, i.e., a new node that wants to be added to a neighborhood in order to contribute to the processing of the IoT data. In this algorithm, each new node n_{new} joins through C which guides n_{new} to a suitable neighborhood.

Specifically, Algorithm 1 works as follows: Initially, there is only one compute node C which is the root node (cf. C in Fig. 2a), and acts as the entry point to the fog computing system (Line 1), as discussed in Section II-A. Upon request, C examines the number of nodes in the neighborhood that contains children. If this neighborhood is empty, then the response of C is empty (Line 3), and this triggers n_{new} to create a new neighborhood, add n_{new} and C (Line 4), and then notify the neighbors (Line 5) about the arrival of n_{new} (in this case, there is only one neighbor, i.e., C). This means that n_{new} is added as the first child of C.

If the neighborhood that contains the children of C is not empty, and the current size is smaller than m, then the response of C contains this neighborhood (Line 6). Then, n_{new} adds the nodes of this neighborhood as neighbors (Line 7), and also notifies these neighbors about the arrival of n_{new} (Line 8) so that they can add n_{new} as a neighbor. This means that n_{new} is added as a child of C to a preexisting neighborhood.

Finally, if the neighborhood which includes the children of C is at capacity, i.e., this neighborhood contains m neighbors, then the response of C contains the address of one of its children n (Line 9), to be used by n_{new} in a new join request (Line 10). In this case, n_{new} requests to join the fog computing system again using n, and the same process repeats until n accepts n_{new} as a child.

In order to decide which child should be selected in a new join request (Line 10), the parent node uses a cyclic counter with values that correspond to the children. This means that each child is selected interchangeably, which results in a hierarchy that grows in breadth before growing in depth. Notably, Fig. 2a shows a fog computing system which includes new compute nodes that have joined the system sequentially, i.e., C is the first, n_2 is the second, n_3 is the third, and so on. Thus, based on the cyclic counter, the first three nodes that join, i.e., n_2, n_3, and n_4 become children of C, and after that, each new node becomes a child of n_2, n_3, and n_4 interchangeably. The reason we do this is that if the number of nodes between each leaf and the cloud is the same, then the processing from the IoT devices can be distributed evenly among the compute nodes of the system. For instance in Fig. 2a, each path from the edge to the cloud includes three compute nodes (e.g., n_5, n_2, and C).

In order to consider proximity in the hierarchical architecture, each new node measures the proximity to existing nodes, and joins the neighborhood which contains the nodes of the closest proximity. In Algorithm 1 for example, instead of having C decide which node should be used for new join requests (Line 10), n_{new} can decide based on proximity. To achieve this, when the neighborhood of children is at capacity (Line 9), n_{new} receives all these children, and decides which one to join (Line 10), after measuring the proximity to each child (e.g., using hop count). Notably, when proximity is considered, instead of sending the IoT data directly upwards the hierarchy, it is possible to send the data to the neighbors on a spanning tree. This can reduce the bandwidth utilization [21]. In Fig. 2a for example, instead of sending data from n_5 to n_2 directly, it has been shown that sending the data to n_2 through n_8 might be more efficient due to the potential difference in the bandwidth capacity of the paths. By following these adaptations it is possible to implement an algorithm which results in a hierarchical architecture with proximity awareness, i.e., HWP (Hierarchical architecture With Proximity). Such an algorithm is realized in Section III for comparison reasons.

C. The Flat Architecture

Similar to the hierarchical, in the flat architecture the cloud is considered to be the initial entry point to the fog computing system, and may also be used as a global point of coordination. However, in contrast to the hierarchical, in the flat architecture any existing compute node of the system can be used as an entry point. As a result, the neighborhoods are formed based on the nodes that are used as entry points, and may consist of nodes with very diverse capacities [21]. Consequently, when the liaisons propagate the IoT data, this data does not

Algorithm 2: FNP

```
1  joinFlat(computeNode n_ep){
2  joinResponse = joinRequest(n_ep)
3  if joinResponse == ∅ then
4  |  H_new.add(this, n_ep)
5  |  H_new.updateNeighbors()
6  else if joinResponse == H then
7  |  H.add(this)
8  |  H.updateNeighbors()
9  end
10 }
```

necessarily head towards the cloud (which is the case in the hierarchical architecture as discussed in Section II-B). Thus, the assumption that more computational resources are found when the IoT data is propagated to other neighborhoods, as discussed in Section II-B, does not hold anymore.

In order to organize the compute nodes of a fog computing system in a flat architecture without proximity awareness, we design FNP (Flat architecture No Proximity). The steps of FNP are shown in Algorithm 2 which explains the algorithm from the perspective of a new compute node that requests to join the fog computing system. In this algorithm, each new compute node n_{new} joins through a node that acts as the entry point n_{ep} to the fog computing system, which can be any preexisting node of the system (initially, it is C).

Upon request (Line 1), n_{ep} examines the neighborhoods that it belongs to. If all of these neighborhoods are at capacity (or n_{ep} does not belong to any neighborhoods), n_{ep} sends back an empty response (Line 3). The empty response triggers n_{new} to create a new neighborhood, to add n_{new} and n_{ep} (Line 4), and to notify n_{ep} about the arrival of n_{new} (Line 5). This means that n_{new} is added as the first neighbor of n_{ep} in a neighborhood that includes only n_{new} and n_{ep}. More nodes can be added to this neighborhood upon request, and until the size of the neighborhood reaches m. If n_{ep} belongs to a neighborhood which is not at capacity, n_{ep} sends back a response that contains this neighborhood (Line 6). Then, n_{new} adds the nodes of this neighborhood as neighbors (Line 7), and notifies the neighbors about the arrival of n_{new} (Line 8). This means that n_{new} is added to a preexisting neighborhood of n_{ep}.

In order to make the compute nodes that follow the flat architecture aware of the proximity of their neighbors, each node needs to take proximity measurements (e.g., using hop count). To achieve this, the following adaptations are required. After n_{new} requests to join, the compute node which acts as the entry point n_{ep}, may send back a response with a neighborhood (Line 6). In case n_{ep} belongs to many neighborhoods, then n_{ep} sends a response which contains all of these neighborhoods. This allows n_{new} to take proximity measurements, and to select the neighborhood that contains the neighbors of the closest proximity. By doing this, an algorithm which results in a flat architecture with proximity awareness can be created, i.e., FWP (Flat architecture With Proximity). We implement FWP in Section III for comparison purposes.

Notably, the layout of a fog computing system created using FNP depends on the nodes that are used as entry points. For instance, in Fig. 2b only the nodes n_4 and C have been used as entry points. This is why all the other nodes are gathered around n_4 and C. Additionally, since new neighborhoods always include n_{ep} and n_{new}, and because n_{new} does not belong to other neighborhoods (since n_{new} is new), only n_{ep} can become a liaison. Thus, not using specific nodes as entry points ensures that these nodes will not become liaisons. This can be useful for scenarios with resource-constrained compute nodes which may have enough computational resources to process the IoT data, but not enough to also propagate the data to other neighborhoods.

III. Evaluation

In order to build hierarchical and flat fog computing systems as discussed in Sections II-B and II-C, we implement a prototype of the proposed algorithms in Java. The source code of the prototype along with the code used for this evaluation, and the produced numerical results, can be found in the project repository [29]. By using this prototype, it is possible to build testbeds with various compute nodes that communicate with each other either hierarchically or based on the flat architecture, as discussed in Section II.

To evaluate the different architectures and produce representative results which apply to the general case, rather than individual deployments, we perform extensive simulations. Since alternative fog computing simulators assume hierarchical organization among the compute nodes [30], such simulators are not suitable for flat architectures. For this reason, along with the prototype, we also build a simulator which we use for this evaluation. This simulator can be used for performing experiments with a configurable number of compute nodes which form both hierarchical and flat fog computing systems.

As discussed in Section II, the examined approaches are: the hierarchical HNP (no proximity) and HWP (with proximity), and the flat FNP (no proximity) and FWP (with proximity). Since all of these approaches were introduced with a neighborhood size that equals four (as shown in Figs. 2a and 2b), we keep the same neighborhood size in order to make the resulting fog computing systems more comprehensible. Nevertheless, different neighborhood sizes have also been examined in preliminary experiments, and exhibit similar behavior.

In order to perform experiments, first we emulate a fog computing environment. To do this, we build a network that consists of multiple connected access points, and resembles the Internet topology, as shown in Fig. 3. Then, we use the nodes of this network as compute nodes. Notably, to take into account the various network nodes of the infrastructure which do not take part in the processing of the data, we consider only the nodes with one arc as compute nodes. The others (i.e., the access points) are considered as nodes of the infrastructure, and are only accounted for when measuring proximity based on hop count.

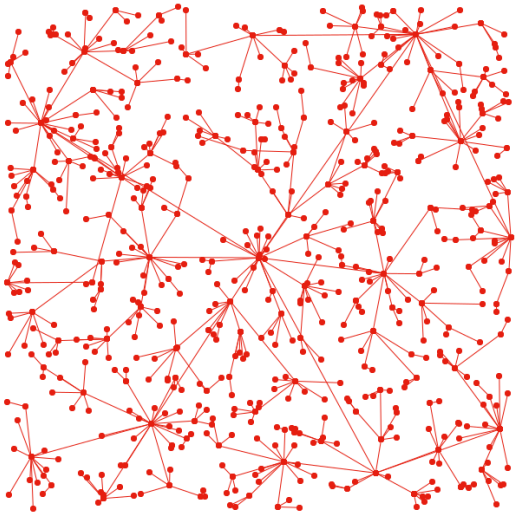


Fig. 3: Example of a generated network with 500 compute nodes.

Based on this setup, we perform 30 experiments with 500 compute nodes for each one of the different approaches (i.e., HNP, FNP, HWP, and FWP). These numbers allow us to capture the general behavior of the system according to the different architectures. For each experiment, we create a randomly generated underlying network (such as the network shown in Fig. 3) using the uniform distribution. The experiments we conduct for this evaluation produce results regarding communication latency, which are presented in Section III-A, and results regarding bandwidth utilization, which are presented in Section III-B. Afterwards, we provide a discussion of these results in Section III-C.

A. *Experimental Results: Communication Latency*

To examine the communication latency in each one of the architectures, we measure the number of hops among neighbors. We assume that the number of hops between two compute nodes is an indicator of proximity, and that proximity can be associated with communication latency. To count the hops, we initiate the transmission of a message from the cloud to its neighbors. Each neighbor that receives this message repeats the transmission to its neighbors until the message is spread epidemically to all the compute nodes. Each time the message is sent to a neighbor, we count the number of hops of the path between the sender and the receiver. For each one of the examined fog computing architectures, we repeat this experiment 30 times in randomly generated networks, and we plot the number of hops of all the paths in Fig. 4. The average and the standard deviation of these values, are shown in Table I.

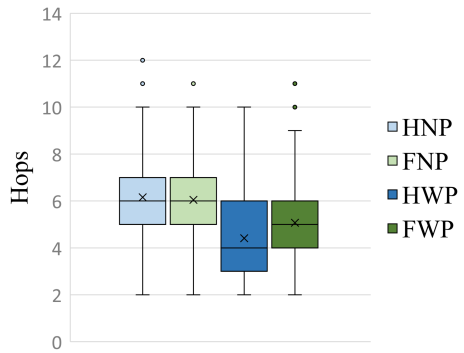


Fig. 4: Proximity of the neighbors (in hops).

The average proximity among the neighbors in HNP is similar to FNP, and is approximately 6 hops. When taking into account the proximity among the compute nodes, the average proximity drops for both the hierarchical and the flat architecture. However, in HWP the average proximity drops to 4.41 hops, while in FWP the average proximity drops to 5.07 hops. This means that considering proximity reduces the communication latency among neighbors by approximately 28% in the hierarchical architecture, and by approximately 16% in the flat architecture. Furthermore, we notice that in HWP the average proximity is approximately 13% less than in FWP.

As indicated by Fig 4, the communication latency in HWP and FWP is reduced compared to HNP and FNP. The reason for this is that, as discussed in Sections II-B and II-C, in HWP and FWP the new compute nodes select neighbors according to proximity measurements. Furthermore, we note that in HWP the communication latency is lower than in FWP. This happens because in the hierarchical architecture, the join request of a new compute node can be propagated downwards the hierarchy, until a neighborhood with nearby nodes is found. In FWP on the other hand, a new compute node examines the neighborhoods around the entry point, rather than exploring other neighborhoods of the system. Thus, the search space in FWP is smaller than in HWP, and the candidate neighborhoods are fewer. This reduces the chances of finding a neighborhood with compute nodes in proximity.

Apart from the proximity among the neighbors, we also measure the proximity of the cloud, which is considered as an indicator of the communication latency required to reach the cloud. This is an important metric because fog computing operates on a cloud-to-thing continuum which means that interactions with the cloud are likely. Such interactions occur mainly when the compute nodes at the edge or close to the edge of the network, do not have enough computational

TABLE I: Average and standard deviation of the proximity values in Fig. 4.

	HNP	FNP	HWP	FWP
Average Value	6.16	6.05	4.41	5.07
Standard Deviation	1.49	1.47	1.69	1.54

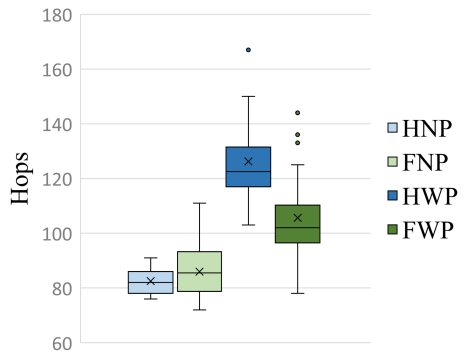


Fig. 5: Proximity of the cloud (in hops).

resources to process the IoT data. To examine this communication latency in each architecture, we count the number of hops between the cloud and the compute node at the edge, which resides the farthest away. To do this, we send a message from the cloud to all the other compute nodes. Then, we count the hops to reach each one of these nodes, and we find the maximum number of hops. This number corresponds to the node which resides the farthest away. We repeat this experiment 30 times in randomly generated networks for each architecture, and we plot the results in Fig. 5. The average and the standard deviation of these values are shown in Table II.

The average proximity of the cloud in HNP is 82.5 hops, and is slightly lower than in FNP (85.93 hops). When considering the proximity among the nodes, the proximity of the cloud in HWP increases to 126.23, which is approximately 35% more than HNP. Finally, when taking into account the proximity of the nodes in FWP, the proximity of the cloud increases to 105.63 which is approximately 19% higher than FNP, and approximately16% lower than HWP.

Surprisingly, according to Fig. 5, the proximity of the cloud increases when taking into account the proximity among the nodes. The reason that this happens is that in architectures that consider proximity, the compute nodes tend to form longer paths to reach the cloud. In Fig. 4, we notice that HWP provides the lowest proximity among neighbors. This is achieved by placing many nodes between the cloud and the edge of the network, which reduces the latency to reach the neighbors. However, this also means that in order to send a message from a compute node at the edge of the network to the cloud, this message has to go through more neighbors. Thus, the path from the edge to the cloud includes more nodes, increasing the communication latency to the cloud.

Moreover, in Fig. 5 we note that the communication latency to reach the cloud in FWP is lower than HWP. The reason for this is that as shown in Fig. 4, HWP manages to place neighbors close to each other, but creates longer paths to the cloud. FWP still reduces the communication latency among the neighbors compared to FNP, but not as much as HWP.

TABLE II: Average and standard deviation of the proximity values in Fig. 5.

	HNP	FNP	HWP	FWP
Average Value	82.5	85.93	126.23	105.63
Standard Deviation	4.3	9.01	13.81	16.27

As a result, the paths are not as long as in HWP, and the communication latency to reach the cloud is lower. Hence, we note that there is a trade-off between low proximity to the neighbors, and low proximity to the cloud.

B. Experimental Results: Bandwidth Utilization

To measure the bandwidth utilization among the different architectures, we perform the following experiment: We send a message from the cloud to all the other nodes, and we count the number of hops traveled until all the nodes receive this message. This is considered as an indicator of bandwidth utilization because each time a message travels from one node to the other, a part of the available bandwidth of the link that connects these two nodes, is used. In HNP and FNP, the message is sent to the neighbors, and each neighbor repeats the transmission until all the nodes have received the message. In HWP and FWP, since each compute node is aware of the proximity to its neighbors, the message is sent to the neighbors on a spanning tree, which has been proposed to reduce bandwidth utilization (as discussed in Section II-B). We repeat this experiment 30 times on randomly generated networks, and we plot the bandwidth utilization of each architecture in Fig. 6. The average and the standard deviation of these values, are shown in Table II.

Fig. 6 shows that the average bandwidth utilization in HNP is similar to FNP, with a number of hops that is slightly higher than 3000 hops. When considering the proximity among the nodes, this number drops significantly. In FWP, the average number of required hops drops to 2529.9 which is approximately 16% less than FNP. In HWP, the average number of hops drops to 2200.37, which means approximately 28% less than HNP, and approximately 13% less than FWP.

The reason for having reduced bandwidth utilization in HWP and FWP, is that taking into account the proximity among the nodes, results in logical links among neighbors, which match the underlying network. This means that when sending a message to a neighbor, this message travels a shorter distance on the underlying network, which leads to less traveled hops, and lower bandwidth utilization. Furthermore, when each node is aware of the proximity of its neighbors, sending messages on a spanning tree may reduce the number of traveled hops required to send messages, even more. However, this is not possible in HNP and FNP because the proximity among the nodes is not considered.

Notably, HWP utilizes significantly less bandwidth than FWP. The reason for this is that this hierarchical architecture, as shown in Fig 4, is more efficient at connecting neighbors

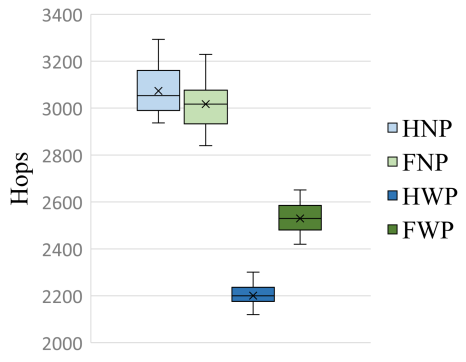


Fig. 6: Bandwidth utilization of sending a message to all the compute nodes (in hops).

based on proximity, and at matching the logical links with the underlying network. FWP still reduces the bandwidth utilization compared to FNP, but not as much as HWP, since the neighbors in FWP are, by average, not as close to each other as in HWP (as shown in Fig 4).

C. Further Discussion

By comparing hierarchical and flat fog computing architectures with and without proximity awareness, we note that each architecture may prove efficient for different use cases. The hierarchical architecture without proximity awareness, can be very efficient for applications that leverage on compute nodes at the edge of the network, but also require frequent communication with the cloud (cf. Fig. 5). Such applications can be related to use cases such as online storage, which may need to execute tasks at the edge (e.g., data compression), but also include resource-demanding tasks (e.g., feature extraction using machine learning). These tasks are preferably executed in the cloud, due to the limited computational resources at the edge [26].

The flat architecture without proximity awareness shares similar properties with the hierarchical (without proximity awareness), and can be useful for similar use cases. However, the proximity to the cloud is slightly increased (cf. Fig. 5), while the bandwidth utilization is slightly decreased (cf. Fig. 6). These properties may make the flat architecture more appropriate for dealing with use cases with larger files which require more bandwidth.

The hierarchical architecture with proximity awareness exhibits the lowest communication latency (cf. Fig. 4) among neighbors compared to all the alternatives, and also the lowest bandwidth utilization (cf. Fig. 6). However, this is accompanied by the highest communication latency to reach the cloud (cf. Fig. 5). These properties make the hierarchical architecture (with proximity awareness) more suitable for critical applications which require low communication latency with the compute nodes at the edge, and do not necessarily need to interact with the cloud. Use cases that can benefit from this, may be, e.g., industrial manufacturing and industrial IoT, because such use cases commonly include applications with stringent latency requirements [31].

TABLE III: Average and standard deviation of the bandwidth utilization values in Fig. 6.

	HNP	FNP	HWP	FWP
Average Value	3073.17	3017.2	2200.37	2529.9
Standard Deviation	98.17	103.61	44.76	60.02

Finally, the flat architecture with proximity awareness can be considered as the general-purpose fog computing architecture, because it provides a balance between low communication latency with low bandwidth utilization. This is supported by all the examined criteria in which the flat architecture (with proximity awareness) provides moderately good results, i.e., low communication latency among the neighbors (cf. Fig. 4), low communication latency with the cloud (cf. Fig. 5), and low bandwidth utilization (cf. Fig. 6).

IV. Related Work

We identify related work both in papers that propose a system model and a fog computing architecture, and in review papers which aim at discussing differences among the already proposed architectures. For this reason, in Section IV-A we present various fog computing architectures from the literature, and in Section IV-B we discuss related review papers.

A. Related Fog Computing Architectures

Fog computing was introduced using the hierarchical architecture [32] although, flat architectures were soon found to be prominent as well. For this reason, related approaches using both architectural styles are discussed below.

Sinaeepourfard et al. [28] propose a hierarchical fog computing architecture for managing the data from the IoT devices. This architecture consists of a cloud compute node at the top, while the other compute nodes which are closer to the IoT devices, are organized in layers below. The number of layers in this architecture depends on the number of participating compute nodes, and increases when more nodes join. In this approach, there are no actual proximity measurements among the nodes. Instead, it is assumed that the latency to reach the cloud is high, whereas the latency to reach the other compute nodes is low. The goal is to reduce the communication latency of processing the IoT data. To achieve this, the data is stored temporarily in the compute nodes of low layers. If the data is not requested, it moves upwards the hierarchy until the cloud is reached. Even though this approach provides a sound architecture for hierarchical fog computing, alternative architectures are not examined. In our work, we examine various fog computing architectures, and we discuss the differences based on quantitative results.

Nguyen et al. [33] present a three-layer hierarchical architecture in which a service provider deploys services on

available compute nodes at the edge of the network. These services are deployed in the proximity of the IoT devices in order to reduce the communication latency. To estimate proximity, the authors propose a mathematical model to calculate the round-trip times of messages. Furthermore, in this work each compute node is able to discover if a service is deployed in neighbor nodes, in order to manage the available computational resources in an efficient manner. Summed up, this approach provides an efficient and robust architecture for computing at the edge of the network. However, the reason that a three-layer hierarchical architecture is chosen in not discussed, and there is no comparison with alternatives. In our work, we design both hierarchical and flat fog computing architectures, and we examine the differences.

Santos et al [34] propose a fog computing architecture for enabling automatic resource discovery for IoT services. In this architecture, the compute nodes communicate with each other without hierarchical control, but by using distributed hash tables (DHTs). This approach does not take into account actual proximity measurements, but instead, it relies on the DHTs for organizing the nodes. DHTs provide node lookup operations whereby the compute nodes can exchange information regarding the placement of the IoT services, e.g., the amount of available computational resources of each node. Thus, the authors propose a flat architecture based on DHTs in order to enable the participating compute nodes to communicate with each other, and to facilitate applications. Notably, the efficiency of this approach is not compared to a hierarchical architecture. In our work, we design both hierarchical and flat architectures, and we discuss efficiency aspects regarding communication latency and bandwidth utilization.

In our former work [21], we propose a flat fog computing architecture in which each compute node communicates with few nodes in proximity. To measure proximity, we propose the use of hop count. Each compute node in this architecture can receive a request for application execution. Upon request, the node distributes the computations of an application among the neighbor compute nodes. Furthermore, we propose a messaging mechanism which sends the data to neighbor nodes on a spanning tree in order to reduce the bandwidth utilization. However, this work does not discuss alternative architectures, which raises questions regarding the efficiency of the proposed mechanisms (e.g., the messaging mechanism) when applied to a hierarchical architecture. In the work at hand, we discuss both hierarchical and flat architectures, and we examine the efficiency of messaging neighbors on each one of these architectures, in order to show the differences.

Notably, various fog computing architectures have been proposed so far. As described in the discussion above, some of these architectures are hierarchical, and make use of layers to organize the compute nodes, whereas others are flat, and do not use layers. Independently of this, in some architectures the notion of proximity is integrated, and the compute nodes consider proximity measurements (e.g., round-trip times or hop count) when communicating with each other. In other architectures, the proximity among the nodes is assumed without any actual measurements. However, to the best of our knowledge, no related work provides a comparison between hierarchical and flat fog computing architectures regarding communication latency and bandwidth utilization. For this reason, the work at hand presents a unified system model which is able to represent both architectures, and conducts an evaluation of these architectures in order to examine the differences.

B. Related Fog Computing Reviews

Varshney and Simmhan [35] provide a taxonomy of approaches for specifying and solving the problem of application placement in compute nodes that span from the cloud to the edge of the network. To this end, the authors present a literature review of fog computing architectures and models, and focus on the placement techniques that have been proposed so far. Thus, this work can benefit developers and researchers that are concerned with developing, designing, and selecting appropriate placement algorithms for fog computing. However, guidelines for selecting an appropriate fog computing architecture, which is the aim of the work at hand, are not provided.

Buyya et al. [36] present a manifesto which discusses various aspects of cloud and fog computing, and aims at identifying challenges, state-of-the-art solutions, and potential limitations. This work discusses different fog computing architectures along with emerging trends and future research directions. Notably, in this work the communication among the compute nodes and the utilized architecture (i.e., networking aspects), are considered as a challenge, and various related works are discussed. However, the differences between using a hierarchical and a flat architecture are not discussed. In our work, we examine these differences based on quantitative results.

Varghese and Buyya [37] discuss various computing architectures and models for decentralizing the cloud, and for computing at the edge of the network. Moreover, the advantages of such architectures are analyzed, and potential future research challenges are discussed. However, this work does not compare the alternative architectures, and does not provide any quantitative results to show the differences. On the contrary, in our work we discuss alternative architectures in fog computing, we implement the different approaches, and we conduct various experiments. Based on the results, we are able to provide guidelines for selecting an appropriate architecture according to the requirements of the final applications in mind.

V. Conclusion

Within this paper, we present and compare hierarchical and flat architectures for fog computing. To this end, first we create a unified system model which is able to represent both of these architectures. Then, we build both hierarchical and flat fog computing systems, and we perform various experiments in order to evaluate the communication latency, and the bandwidth utilization of each architecture. Finally, according to the results which quantify the differences between hierarchical and flat architectures for fog computing, we provide a discussion

of these results, which can be used as guidelines for selecting an appropriate architecture based on the target use case.

A promising research direction on this topic, is to examine the reasons for which each architecture performs differently, and to consider combining different approaches in order to tailor the performance of a fog computing system to the target applications. To achieve this, hybrid architectures which use different approaches for the different parts of the system, may have great potential.

REFERENCES

[1] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, "A survey on application layer protocols for the internet of things," *Transaction on IoT and Cloud Computing*, vol. 3, no. 1, pp. 11–17, 2015.

[2] B. K. Al-Shammari, N. Al-Aboody, and H. S. Al-Raweshidy, "Iot traffic management and integration in the qos supported network," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 352–370, 2017.

[3] T. Anagnostopoulos, A. Zaslavsy, A. Medvedev, and S. Khoruzhnicov, "Top–k query based dynamic scheduling for iot-enabled smart city waste collection," in *International Conference on Mobile Data Management (MDM)*, vol. 2, pp. 50–55, IEEE, 2015.

[4] Y. Li, A.-C. Orgerie, I. Rodero, B. L. Amersho, M. Parashar, and J.-M. Menaud, "End-to-end energy models for edge cloud-based iot platforms: Application to data stream analysis in iot," *Future Generation Computer Systems*, vol. 87, pp. 667–678, 2018.

[5] P. Rathore, A. S. Rao, S. Rajasegarar, E. Vanz, J. Gubbi, and M. Palaniswami, "Real-time urban microclimate analysis using internet of things," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 500–511, 2017.

[6] A. Shukla and Y. Simmhan, "Toward reliable and rapid elasticity for streaming dataflows on clouds," in *International Conference on Distributed Computing Systems (ICDCS)*, pp. 1096–1106, IEEE, 2018.

[7] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[8] D. N. Jha, P. Michalak, Z. Wen, P. Watson, and R. Ranjan, "Multi-objective deployment of data analysis operations in heterogeneous iot infrastructure," *IEEE Transactions on Industrial Informatics*, pp. 1–11, 2019.

[9] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 115–124, IEEE, 2017.

[10] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale iot data analytics for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 677–686, 2017.

[11] J. Hasenburg, M. Grambow, and D. Bermbach, "Towards a replication service for data-intensive fog applications," in *Symposium on Applied Computing (SAC)*, pp. 1–4, ACM, 2020.

[12] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya, "Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 274–283, 2019.

[13] M. Satyanarayanan, "The emergence of edge computing," *IEEE Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[14] J. McChesney, N. Wang, A. Tanwer, E. de Lara, and B. Varghese, "Defog: fog computing benchmarks," in *Symposium on Edge Computing (SEC)*, pp. 47–58, ACM, 2019.

[15] A. Brogi, S. Forti, and A. Ibrahim, "Predictive analysis to support fog application deployment," in *Fog and edge computing: principles and paradigms*, pp. 191–222, Wiley, 2019.

[16] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[17] O. Skarlat, V. Karagiannis, T. Rausch, K. Bachmann, and S. Schulte, "A framework for optimization, service placement, and runtime operation in the fog," in *International Conference on Utility and Cloud Computing (UCC)*, pp. 164–173, IEEE, 2018.

[18] N. Maleki, M. Loni, M. Daneshtalab, M. Conti, and H. Fotouhi, "Sofa: A spark-oriented fog architecture," in *Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 1, pp. 2792–2799, IEEE, 2019.

[19] T. Zhang, J. Jin, X. Zheng, and Y. Yang, "Rate adaptive fog service platform for heterogeneous iot applications," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 176–188, 2020.

[20] A. R. Zamani, M. Zou, J. Diaz-Montes, I. Petri, O. Rana, and M. Parashar, "A computational model to support in-network data analysis in federated ecosystems," *Future Generation Computer Systems*, vol. 80, pp. 342–354, 2018.

[21] V. Karagiannis, S. Schulte, J. Leitão, and N. Preguiça, "Enabling fog computing using self-organizing compute nodes," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2019.

[22] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 36–40, ACM, 2019.

[23] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar, "Mobility-aware application scheduling in fog computing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 26–35, 2017.

[24] V. Karagiannis and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement," in *International Conference on Network and Service Management (CNSM)*, pp. 1–5, IEEE, 2017.

[25] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Transactions on Internet Technology (TOIT)*, vol. 19, no. 1, pp. 1–21, 2018.

[26] R. Ranjan, O. Rana, S. Nepal, M. Yousif, P. James, Z. Wen, S. Barr, P. Watson, P. P. Jayaraman, D. Georgakopoulos, M. Villari, M. Fazio, S. Garg, R. Buyya, L. Wang, A. Y. Zomaya, and S. Dustdar, "The next grand challenges: Integrating the internet of things and data science," *IEEE Cloud Computing*, vol. 5, no. 3, pp. 12–26, 2018.

[27] N. Auluck, O. Rana, S. Nepal, A. Jones, and A. Singh, "Scheduling real time security aware tasks in fog networks," *IEEE Transactions on Services Computing*, pp. 1–14, 2019.

[28] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, and E. Marin-Tordera, "Data preservation through fog-to-cloud (f2c) data management in smart cities," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–9, IEEE, 2018.

[29] "Project repository," in *www.bitbucket.org/BasilKaragiannis/sonproject/*. Accessed online: 9 Feb. 2020.

[30] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[31] S. Chen, Y. Zheng, K. Wang, and W. Lu, "Delay guaranteed energy-efficient computation offloading for industrial iot in fog computing," in *International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2019.

[32] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Workshop on Mobile Cloud Computing (MCC)*, pp. 13–16, ACM, 2012.

[33] T.-D. Nguyen, E.-N. Huh, and M. Jo, "Decentralized and revised content-centric networking-based service deployment and discovery platform in mobile edge computing for iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4162–4175, 2019.

[34] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Towards dynamic fog resource provisioning for smart city applications," in *International Conference on Network and Service Management (CNSM)*, pp. 290–294, IEEE, 2018.

[35] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," *Software: Practice and Experience*, pp. 1–38, 2019.

[36] R. Buyya, S. N. Srirama, G. Casale, R. Calheiros, Y. Simmhan, B. Varghese, E. Gelenbe, B. Javadi, L. M. Vaquero, M. A. Netto, *et al.*, "A manifesto for future generation cloud computing: research directions for the next decade," *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–38, 2018.

[37] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, vol. 79, pp. 849–861, 2018.