Accepted at: 3rd IEEE International Conference on Fog and Edge Computing (ICFEC 2019), pp. 1–10

# Enabling Fog Computing using Self-Organizing Compute Nodes

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

João Leitão, Nuno Preguiça
*NOVA LINCS,* Universidade Nova de Lisboa, Portugal
{jc.leitao, nuno.preguica}@fct.unl.pt

***Abstract*—The emergence of fog computing has led to the design of multi-layer fog computing models which are organized hierarchically. These models commonly dictate the hierarchical structure to all the participating compute nodes. However, organizing the compute nodes by adding customized connections that do not abide by the hierarchical approach, may result in improved performance due to the network's properties i.e., latency or bandwidth between the nodes. For this reason, in this paper we propose an alternative to the hierarchical approach, which is the self-organizing compute nodes. These nodes organize themselves into a flat model which leverages on the network's properties to provide improved performance. The results of the evaluation show that this approach reduces bandwidth utilization (∼30%) by using optimized messaging instead of direct messaging. Furthermore, we show that following a flat model, enables the design of mechanisms for fault tolerance which has been mostly neglected in existing hierarchical models.**

***Index Terms*—Edge computing, Peer to Peer**

## I. Introduction

The advent of the Internet of Things (IoT) has initiated an era with applications that connect the physical world to the cloud [1], [2]. To facilitate these applications, a new computing paradigm has emerged, which is known as fog computing and extends the cloud to the edge of the network [3]. The compute resources in fog computing are commonly organized hierarchically. In the hierarchy, the compute nodes of the cloud occupy the top layer, the compute nodes of the network (i.e., routers, switches and access points) reside in the middle and the compute nodes with sensing and actuating features reside at the bottom [4]. In such hierarchical models, typically, computation requests originate from the bottom and travel upwards the hierarchy until they reach a compute node with enough resources to execute them [5].

However, these models dictate the hierarchical structure to all the participating compute nodes. This means that computation requests always follow the paths of the hierarchy, even if customized connections (that break the hierarchy) enhance performance [6], i.e., reduce latency or improve bandwidth utilization. Based on this observation, we propose an alternative to the hierarchical approach, which is a flat model built using the introduced self-organizing compute nodes (SONs). SONs organize themselves based on a configurable proximity measure (e.g., hop count, latency, bandwidth, etc.) and then communicate with each other on paths that minimize the distance.

This work was funded by the European projects: H2020 LightKone (grant agreement: 732505) and H2020 FORA (grant agreement: 764785).

Moreover, to account for fault tolerance which has been mostly neglected in the context of fog and edge computing [7], [8], we show that the flat model is suitable for implementing tolerance to node failures. In the flat model, nodes can be designed to react to failure by creating additional links to other nodes which are still responsive [9]. In addition, the flat model resolves the single point of failure which is still a drawback of the hierarchical approach [10].

In order to make our approach applicable to fog-based environments, we design SONs considering the identified requirements of fog computing (cf. Section III-A). According to these requirements, we implement SONs for providing a platform for general purpose computations, which exploits all the available compute nodes of the fog. Each SON maintains connectivity information about nearby nodes and share its compute resources in order to execute applications submitted by a user. Therefore, the SONs organize themselves into a flat model and are able to receive applications and execute them as a collective.

The contributions of this work are: We identify the essential requirements for implementing fog computing that follows a flat model and we implement SONs that meet these requirements. Specifically, the SONs integrate novel mechanisms for adding new distributed compute nodes to a network, for achieving fault tolerance and for applying intelligent messaging based on optimization logic.

The rest of this paper abides by the following structure: Section II contains a discussion of related work. Afterwards, Section III analyses the design of SONs for fog computing. Consequently, Section IV describes a prototype implementation and presents an evaluation which focuses on network performance by examining the following aspects: scalability (Section IV-B), fault tolerance (Section IV-C) and intelligent messaging (Section IV-D). Finally, Section V concludes the paper and provides future research directions.

## II. Related Work

Notable advances in the field of fog and edge computing commonly follow a hierarchical layered approach [11]. For instance, Bellavista et al. [12] propose a three-layer architecture for service execution at the edge and Deng et al. [13] discuss hierarchical service provisioning in distributed edges.

In contrast, the work at hand is inspired by self-organizing systems which do not use layers. For this reason, in the following paragraphs we discuss approaches which apply the concept of self-organization to address computing at the edge of the network.

Prazeres and Serrano [14] present SOFT-IoT, a platform for providing interoperability in fog computing environments. This work is motivated by the variety of different platforms that originate from the IoT and the lack of one global fog ecosystem. In this platform, intense data processing occurs in the cloud, while IoT data can also be processed locally. The system model consists of three main entities which are: the devices, the gateways and the servers. The servers integrate a self-organizing monitoring service. This service is responsible for the self-organization of the platform and aids in the operation of other services related to deployment, recovery and management. However, the scope of this work is the description of the model and thus, no concrete self-organizing mechanisms are described. In our work, we design, implement and evaluate such mechanisms.

Aditya and Figueiredo [15] propose Frugal, a distributed mechanism that uses online social networks to create a self-organizing overlay network which can be used as a messaging system for fog computing. The motivation of this work stems from the challenge to manage and coordinate numerous fog devices such as sensors, actuators, personal computers and cloudlets. This study highlights that device to device communication is a necessary enabler for fog computing and that decentralized overlay networks can provide scalable solutions for interconnecting the participating devices. For this reason, the authors design a mechanism to construct an overlay network from social graphs by having the devices exchange information about their neighbors. Notably, this approach assumes knowledge of social neighbors to build the network. In contrast, we design mechanisms that do not depend on such information.

Tato et al. [16] present a self-organizing overlay network named Koala, which addresses the decentralization of cloud computing. The Koala overlay lowers the protocol overhead by eliminating redundant maintenance. This is done by utilizing the application traffic to refine the network instead of costly periodic mechanisms. Moreover, this overlay network provides locality awareness by selecting each routing hop based on a trade-off between hop count and latency. The participating nodes of Koala are organized into a ring rather than hierarchically. However, Koala targets explicitly an environment with geographically distributed mini datacenters. In our work, we also consider resource heterogeneous compute nodes.

Ali et al. [17] address the connectivity of fog networks by tackling the problem of connecting a large number of IoT devices with distributed fog nodes. The model of this work assumes that each fog node can serve many IoT devices whereas, each IoT device is assigned to one fog node. To connect the involved resources and form fog networks, an optimization problem is formulated, which aims at minimizing latency subject to the maximum workload capacity of the fog nodes. For solving this problem, a self-organizing algorithm is proposed. According to this algorithm, IoT devices and fog nodes have preferences towards each other and each IoT device tries to connect with the most preferred fog node. Since the scope of this work is to form fog networks, fault tolerance mechanisms are not discussed. In our work, we present self-organizing compute nodes but also, we implement a mechanism to cope with potential node failures.

Finally, Song et al. [18] propose opportunistic data sharing which allows devices in edge computing environments to discover each other and retrieve required data. The authors motivate this work by noting that edge environments hold unique conditions for implementing data sharing. Each device may contain data and might be in need of certain different data, but it is not known a priori what kind of data each device carries or needs. Since the participating devices are in the proximity of each other for a limited amount of time, it is during this period that they have to organize themselves, form connections and exchange data. For this reason, the authors design mechanisms for discovering all the data within nearby edge devices and for forming different connections for different chunks of data while choosing the devices of the closest proximity in order to minimize the overhead. This approach targets data exchange among nearby nodes which form small scale networks while the mechanisms we propose, aim at being applicable to large scale network as well.

## III. Self-Organizing Compute Nodes

In this section, we introduce the SONs for computing at the edge of the network. To do so, we first identify essential requirements for fog computing in Section III-A and we provide basic definitions in Section III-B. Afterwards in Sections III-C to III-H, we design SONs which explicitly satisfy these requirements.

### A. Fog Computing Requirements

In order to design and implement SONs for fog computing, in the following, we identify the requirements which have to be met:

**Proximity awareness**. Fog computing aims at improving the performance of latency-sensitive applications [19]. To achieve this, proximity awareness is necessary because distributing information among nearby nodes improves the communication efficiency [20]. Even though connecting nodes based on proximity has been achieved in self-organizing systems, in fog computing, the proximity information can be used further for ensuring that the application demands are met (e.g., due to the stringent latency requirements).

**Fault tolerance**. Maintaining connectivity in fog computing can be difficult. When nearby compute nodes become unresponsive, moving the computations to the cloud may impact the performance of the applications [21]. Hence, mechanisms for achieving fault tolerance become challenging.

**Intelligent messaging**. Since the compute nodes in fog computing are assumed to be proximity-aware, customized messaging techniques can further improve the communication

among the participating nodes and achieve better utilization of the available resources [22].

**Scalability**. Fog computing extends the cloud to the edge of the network [3]. Coordinating all the involved resources requires special attention for the communication mechanisms which are expected to scale massively [20].

**Heterogeneity**. Fog computing includes a variety of resource heterogeneous compute nodes which will be deployed on a variety of environments [3]. For this reason, supporting node heterogeneity becomes essential.

**Resource sharing**. A distinguishing characteristic of fog computing is that it enables latency-sensitive computations to be executed by a group of shared resources [23]. Therefore, mechanisms for sharing resources and for invoking the execution of tasks in nearby compute nodes become crucial.

### B. Definitions

This section presents basic definitions which are used hereinafter. The network of all the participating compute nodes is referred to as a *fog network* (e.g., Fig. 1). A compute node that implements logic to aid in the process of adding new compute nodes to the fog network as well as to cooperate with them and share resources is referred to as a *SON*. To enable the execution of applications by multiple compute nodes which operate as a collective, SONs form *groups* (cf. the groups shown in Fig. 1). Two SONs that belongs to the same group are referred to as *neighbors*. As shown in Fig. 1, a SON may belong to different groups at the same time. Hence, the neighbors of one SON do not necessarily belong to the same group. For instance, not all neighbors of the cloud compute nodes in Fig. 1 belong to the same group. Each group consists of a finite number of SONs. SONs that belong to the same group share information about each other through the *group graphs*.

A group graph is a weighted and complete graph whereby the vertices represent the SONs and the weights of the arcs represent a proximity measure (e.g., hop count, latency, combinations thereof, etc.). The vertices of the group graphs maintain properties of their corresponding SONs. Such properties are: resource capacities and resource availability as well as hosted services and their requirements. These are used for sharing computations, as discussed later in Section III-D. In addition, each SON stores within the group graph a list of its neighbors which are used for fault tolerance, as explained later in Section III-E. The SONs of the same group maintain the same information in their group graph. When a SON belongs to many groups, all the respective groups graphs are stored within the SON. The group graphs are updated using a data store (for each group) which is replicated in all the SONs of a group. The data store synchronizes based on events.

For simplicity, in this work we assume that the number of SONs in each group is bounded by a maximum group size parameter and that the number of groups that a SON belongs to is not bounded. However, each SON can configure these values based on its resource capacities. In this case, the *max group size* refers to the number of SONs that can belong to the same group. This parameter is constrained by the SONs' processing

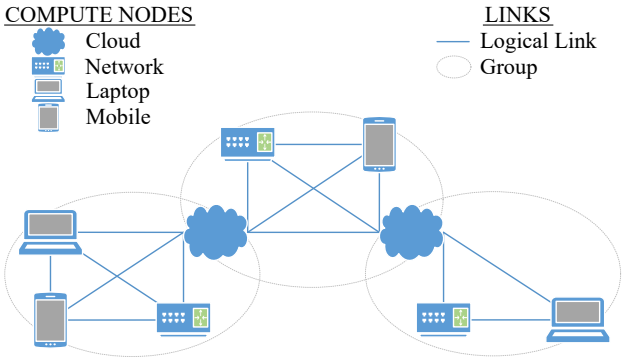



Fig. 1: An example of a fog network consisting of nine self-organizing compute nodes which form three groups.

capacity (CPU) because in order to process the group graphs, complex algorithms may be necessary (e.g., shortest path). The *max number of groups* parameter refers to the number of groups that a SON can be a member of. This parameter is bounded by the storage capacity because for each group that a SON belongs to, the related information of the group graph mush be kept in the local data store.

Regarding application deployment, we use a model which targets the deployment of applications on distributed compute nodes which are represented by a graph [24]. Each application consists of one or more *services*. Each service is an executable file which is deployed independently to the others but potentially interacts with them. The services may have *requirements* [25]. These requirements can be related to hardware (e.g., CPU, memory), compute nodes (e.g., a service requires to be deployed on a specific compute node), services (e.g., a service needs to be deployed at the same compute node as another service) or network (e.g., the latency value between two services needs to be less/more than a threshold). For the application to be executed, all the services must be deployed.

### C. Proximity Awareness

In this section, we present how SONs organize themselves with proximity awareness. To do so, we assume a fog network of one SON and we describe the process of adding new SONs to the network. Each time a new SON is added, its proximity to other SONs is measured and integrated in the group graph. This information can be used for ensuring that the application requirements are satisfied when distributing the services in the fog network, as discussed later in Section III-D. The process that a new SON has to follow in order to be added to a fog network, is presented below and described in Algorithm 1 which follows the notation of Table I.

*1) Adding a New SON:* Initially, a new SON $S_{new}$ is added by a contact SON $S_{con}$ which already belongs to the fog network. Thus, in the case that $S_{new}$ requests to be added to a previously formed fog network, the address of $S_{con}$ is assumed to be known. In case $S_{new}$ wants to start the formation of a new fog network, $S_{con}$ represents another SON which does not belong to any fog networks. In any case, $S_{new}$ is assumed to know the address of $S_{con}$ a priori.

TABLE I: The notation used in Algorithm 1.

| Symbol | Description |
|---|---|
| $S_{new}$ | A new SON to be added to the fog network. |
| $S_{con}$ | A previously added SON used as contact. |
| $S_{clo}$ | The SON in $LG_{S_{con}}$ closest to $S_{new}$. |
| $S_{far}$ | The most distant SON in $G_{S_{clo}}$. |
| $G_{new}$ | A new group for adding $S_{new}$ and $S_{con}$. |
| $G_{S_{clo}}$ | The group in $LG_{S_{con}}$ that contains $S_{clo}$. |
| $LG_{S_{con}}$ | The list of groups that $S_{con}$ belongs to. |

**Algorithm 1:** Process of adding a new SON

```
1  int maxGroupSize                          // maximum group size
2  SON S_con
3  List LG_{S_con}
4  SON S_clo
5  SON S_far
6  GroupGraph G_{S_clo}
7  boolean flag=false                        // true when S_new is added
8  LG_{S_con} = requestLG_{S_con}(S_con)
9  LG_{S_con}.findAllSonDistances()
10 for i = 1 to LG_{S_con}.size() do
11     S_clo = LG_{S_con}.findClosestSon()
12     G_{S_clo} = S_clo.getGroupGraph()
13     G_{S_clo}.add(this)                   // this = S_new
14     G_{S_clo}.updateAllSons()
15     if G_{S_clo}.size() > maxGroupSize then
16         S_far = G_{S_clo}.findMostDistantSon()
17         G_{S_clo}.remove(S_far)
18         if this == S_far then
19             LG_{S_con}.remove(G_{S_clo});
20         else
21             flag = true
22             break
23         end
24     else
25         flag = true
26         break
27     end
28 end
29 if flag == false then
30     GroupGraph G_new
31     G_new.addSons(this, S_con)
32     G_new.addAllSonDistances()
33     G_new.updateAllSons()
34 end
```

$S_{new}$ requests to be added through $S_{con}$ which responds with the list of the group graphs $LG_{S_{con}}$ it belongs to (line 8). Then, $S_{new}$ makes a system call to traceroute (a command for measuring network paths) in order to estimate the proximity (using hop count and latency) of all the SONs in $LG_{S_{con}}$ (line 9) and selects the group $G_{S_{clo}}$ that the closest SON $S_{clo}$ (based on the proximity measure) belongs to (line 12). Next, $S_{new}$ updates $G_{S_{clo}}$ by adding itself and the distances to the other group SONs (line 13) and shares the updated graph with the SONs of $G_{S_{clo}}$ (line 14). Each time a $S_{new}$ is added to $G_{S_{clo}}$, all the SONs of $G_{S_{clo}}$ check the group size by counting the number of vertices in $G_{S_{clo}}$ (line 15). In case the size is greater than the maximum group size parameter, all SONs examine $G_{S_{clo}}$ locally (without network interactions) and find the SON $S_{far}$ that resides the farthest away from the rest (line 16). To define a metric for finding $S_{far}$ , we denote the SONs of a group with N members as $S_1$, $S_2$, ... , $S_N$ and the weight of the arc that connects two SONs $S_i$ and $S_j$ as $W_{S_i,S_j}$. The most distant SON $S_{far}$ is the one for which:

$$\sum_{i=1}^{N} W_{S_{far},S_i} = max\left\{\sum_{i=1}^{N} W_{S_1,S_i}, ..., \sum_{i=1}^{N} WS_N, S_i\right\}$$

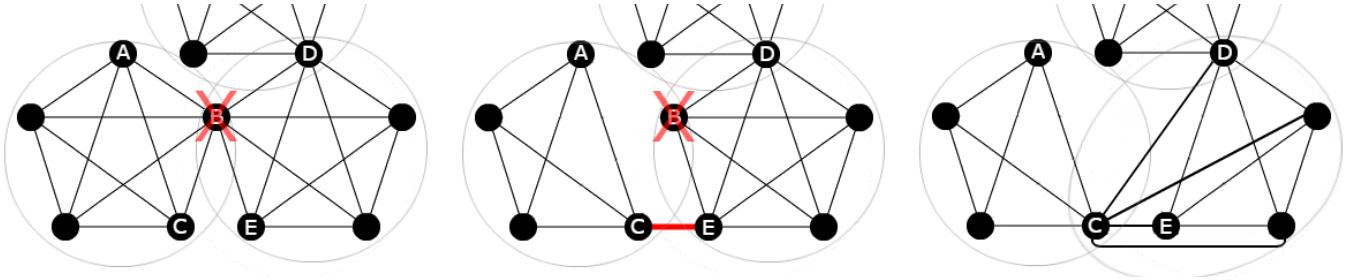
i.e., the SON for which the sum of the weights of the arcs that connect it to the rest of the SONs in the group is the maximum among all the SONs of the group. If there are multiple candidates (i.e., equal sum of weights) for becoming $S_{far}$, we choose the first occurrence which is the same in all the group SONs since their groups graphs are identical (replicated). $S_{far}$ is then removed from $G_{S_{clo}}$ (line 17), i.e., leaves the group and rejoins the fog network using the same process of Algorithm 1 and $S_{new}$ as contact. In case $S_{new} == S_{far}$ (line 18), $G_{S_{clo}}$ is removed from $LG_{S_{con}}$ (line 19) and $S_{new}$ joins the group of the closest SON among the SONs that remain in $LG_{S_{con}}$. The same operation continues for as long as $S_{new}$ leaves $G_{S_{clo}}$ for being the most distant and until all groups $LG_{S_{con}}$ are examined (line 28). If $S_{new}$ != $S_{far}$ (line 20) or if the size of $G_{S_{clo}}$ is smaller or equal to the maximum group size after $S_{new}$ is added (line 24), then $S_{new}$ has joined the fog network as member of $G_{S_{clo}}$.

*2) Creating a New Group:* When all the groups of $S_{con}$ are at capacity and $S_{new} == S_{far}$ in all groups of $LG_{S_{con}}$ (line 29), a new group is created to accommodate $S_{new}$. Specifically, in this case $S_{new}$ creates a new group graph $G_{new}$ (line 30) containing only itself (i.e., $S_{new}$), the $S_{con}$ and the arc between them, and shares $G_{new}$ with $S_{con}$ (line 33). Other SONs can be added to $G_{new}$ upon request until the maximum group size is reached.

Notably, when replacing the most distant nodes with new ones of closer proximity, the disposition of the fog network improves because groups tend to contain SONs that reside close to each other (based on the proximity measure). The decision to use $S_{new}$ as contact for $S_{far}$ enables $S_{new}$ to be added to the group of $LG_{S_{con}}$ which contains $S_{clo}$ without rearranging other groups of $LG_{S_{con}}$. The alternative to use $S_{con}$ as contact for $S_{far}$ may result in a ripple effect that rearranges all the groups of $LG_{S_{con}}$. This effect occurs because $S_{far}$ selects one of the groups of $LG_{S_{con}}$ in contrast to the former case, in which $S_{far}$ selects one of the groups of $S_{new}$ (which is new and thus, it does not belong to other groups). Even though this ripple may improve the disposition of the fog network, it also increases the overhead of adding new SONs. In this work, we avoid this ripple effect in order to achieve higher scalability, as discussed later in Section III-H.

### *D. Resource Sharing*

After building a fog network using the mechanism of Section III-C, a user can submit a request for application execution to any of the participating SONs. This request includes *i*) link addresses of repositories for downloading the executable services (or the services) and *ii*) the requirements of the services. After submitting a request to a SON $S_{app}$




(a) Step 1: C discovers the unresponsive B. (b) Step 2: C sends join-request to E. (c) Step 3: C is added and B is removed.

Fig. 2: Example of the fault tolerance mechanism when SON C discovers the unresponsive SON B.

(i.e., the SON to which the user submitted the request), $S_{app}$ examines the groups it belongs to in order to find if one of these groups can meet the application requirements. This is done locally (without network interactions) using the information stored in the group graphs. If one of the groups can host the application successfully, $S_{app}$ shares the services and their requirements with the selected group and initiates the application execution. The services are instantiated in the SONs using system calls. If none of the groups that $S_{app}$ belongs to can host the application, $S_{app}$ forwards the request for application execution to another SON $S_{app2}$. $S_{app2}$ performs the same procedure but on different groups (i.e., the groups it belongs to).

Due to the way SONs are organized in groups, even though $S_{app}$ and $S_{app2}$ belong to the same group, they may also belong to different groups. Therefore, a request for application execution is examined by different groups when being forwarded to SONs of the same group. For instance, consider a request that is originally submitted to a network SON in Fig. 1 and is examined for the corresponding group. By being forwarded to the cloud SON, it can be examined by a second group because the cloud SON belongs to two groups. This way, requests spread in an epidemic manner within the fog network. Concrete optimization heuristics for placing services on resources represented by a graph based on the application requirements, have been discussed in our previous work [24].

### *E. Fault Tolerance*

According to the way that new SONs are added, fog networks follow a flat model in which different groups are connected through common SONs (cf. Fig. 1). In this model, if a SON $S_{unr}$ which belongs to only one group becomes unresponsive, the fog network remains connected. However, if $S_{unr}$ belongs to other groups as well, the fog network is divided into a number of disjoint networks that equal to the number of the groups that $S_{unr}$ belongs to. In this case, these disjoint networks operate independently from each other and the SONs of each network cannot collaborate with the rest of the fog. Therefore, a fault tolerance mechanism for maintaining connectivity for the case that one of the participating SONs fails, disconnects or departs from the fog network, is essential.

The fault tolerance mechanism we design relies on the application traffic to detect unresponsive SONs. When the services of $S_{unr}$ do not respond to the services of other SONs, $S_{unr}$ is detected. Then, the responsive SON $S_{res}$ that detected $S_{unr}$ triggers a recovery process. According to this process, $S_{res}$ requests to join each one of the groups that $S_{unr}$ belongs to in order to maintain the connectivity with all the disjoint networks of the fog. To do so, $S_{res}$ requests to be added to the fog network again through a responsive neighbor of $S_{unr}$. The neighbors of $S_{unr}$ are known through the group graphs. Fig. 2 shows an example of the recovery process in three steps.

To maintain normal operation of the applications, when $S_{unr}$ is detected, $S_{res}$ examines the group graph to check if any services are deployed on $S_{unr}$. In positive case, $S_{res}$ triggers application redeployment. During this process, $S_{res}$ examines if redeploying only the services of $S_{unr}$ on other SONs of the same group, keeps the application requirements satisfied. If it does, only the services of $S_{unr}$ are redeployed. Otherwise, the application follows the initial submission process, as described in Section III-D.

By using this fault tolerance mechanism, unresponsive nodes (if any) are detected and removed using the application traffic. Thus, assuming normal operation of an application that requires the SONs to communicate frequently, the fog network recovers from failures regularly. Additionally, if $S_{unr}$ disconnects and reconnects to the fog network without having been missed (i.e., no message was addressed to $S_{unr}$ while being offline), no further actions are required since connectivity remains the same. This is especially useful in unstable networks (e.g., with wireless or mobile devices) which are prone to momentary loss of connection.

### *F. Intelligent Messaging*

Since each SON maintains a weighted and complete graph representation of the groups it belongs to, it is possible to design intelligent messaging for improving the communication efficiency. In this work, we claim that instead of messaging multiple SONs independently with the same message, messaging neighbors with the command to forward the message towards the destination SONs, may improve bandwidth utilization. To support this claim we formulate the following optimization problem.

Given any group graph $G = (V, E)$ with $V$ being the set of the vertices corresponding to the SONs, $E = \{e_1, e_2, ...\}$ being the set of the arcs and $W_e$ being the weight of the arc $e$ corresponding to hops. In $G$, assume a SON $S$ that wants to send a message to a set $R$ of SONs within $G$. In this scenario, let $U$ be a set of all the trees that start from the vertex which corresponds to $S$ and reach the vertices that correspond to the SONs of $R$ and the cost function:

$$c(T) = \sum_{e \in T} W_e$$

$$\text{with} \quad T \in U \quad \text{and} \quad U \subseteq E$$

The cost function represents the bandwidth utilization of sending a message from $S$ to all the SONs of $R$ as the summary of hops that have to be traveled until the message is sent. The optimization goal is to send the message with minimum cost, i.e.,

$$min\,\{c(T) : T \in U\}$$

Based on this goal, when $S$ sends a message within the same group, it sends this message on a tree to/through other SONs in order to minimize the cost. We use this cost function to reduce bandwidth utilization within groups in order to improve bandwidth management in fog networks. The formulation of this problem is identical to the formulation of the minimum spanning tree problem (of P complexity) with known solutions provided by the algorithms of Prism and Kruskal [26].

The reason we focus on bandwidth utilization is that fog computing intends to handle massive amounts of IoT data which may cause bottlenecks [23]. However, by altering the proximity measure, different optimization logic is also possible. For instance, assuming that the proximity measure corresponds to throughput, processing the group graph may show that messaging a SON through an intermediate node results in a path with higher throughput [27].

Notably, according to the proximity measure used to enumerate the arcs, the paths of the graph may not be bidirectional. This is the case when the cost to travel from SON A to SON B is different than from B to A. In this case, the group graphs should represent these paths with separate directed arcs and their respective weights.

Based on intelligent messaging, we also design queries which are crucial to IoT applications due to the growing number of devices [8]. Traditional querying occurs through flooding which generates a significant number of duplicate messages [9]. In our proposed approach, due to the way new SONs are added, fog networks maintain an important property which is that there are no circles connecting different groups. This makes it possible to design a querying mechanism which spreads queries epidemically without generating duplicate messages. This mechanism can be initiated by any participating SON using the following process.

First, a query is sent to all the neighbors of the sender SON. Then, each of the receivers examines the query locally and forwards it to all its neighbors, apart from the neighbors of the group from which the message was received. This process continues until no transmission is pending by which time, all SONs have received the query exactly one time. The SONs that match the query respond to the sender SON. Since a fog network can scale massively or because the sender is interested only in nearby SONs, queries can be initiated with an expiration value based on groups traveled.

### G. Heterogeneity

To address SONs with diverse resource capacities, we define the max group size (based on CPU) and the max number of groups (based on memory) in Section III-B. These two parameters can be configured individually by each SON and be taken into account during the construction of the fog network. This ensures that each SON has enough resources for the required control operations, i.e., to store the group graphs, to implement intelligent messaging, etc.

In addition, we foresee the participation of resource constrained nodes (e.g., small devices with microcontrollers connected to sensors and actuators) that cannot implement the mechanisms discussed within this work. Such nodes can still join, but through a SON that acts as a gateway to/from the fog network. This way, all the SONs of the fog network can access values from the sensors and activate/deactivate actuators according to the logic of the respective applications. In the northbound interface of a SON that acts as a gateway, there is a SON API whereas in the southbound interface, the SON acts as an access point to wired/wireless resource constrained devices. Alternatively, the southbound interface may lead to sensor networks while implementing the corresponding API.

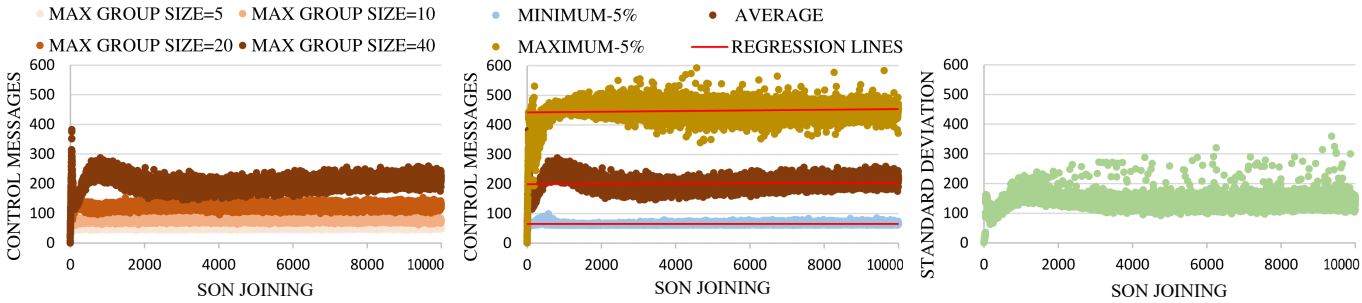
### H. Scalability

Decentralized protocols are designed to scale by not relying on global knowledge of the system [28]. For this reason, SONs do not store information of the whole fog network. Instead, SONs maintain only the group graphs which store information about neighbors. The number of neighbors of each SON is bounded by the max group size and the max number of groups. Thus, the information that is stored within each SON, is not affected by the size of the fog network. Additionally, control operations such as adding new SONs to the fog network and recovering from failure, affect only neighbors while the rest of the fog network remains stable.

### I. Improvements

In this section, we describe tweaks that improve performance by avoiding certain situations that may arise. When using intelligent messaging, messages are sent on a minimum spanning tree. However, there can be many minimum spanning trees starting from the sender node. In this case, we choose the tree that bears minimum latency. Latency is measured as the number of hops in the path of the message that is received by the last SON.

To implement fault tolerance, SONs store their neighbors in the group graphs. This information can be useful to contact nodes for redirecting join-requests to groups with few SONs




(a) Number of exchanged control messages for each new SON that joins the fog network (for different group sizes).

(b) Minimum, average and maximum number of exchanged control messages for each new SON that joins the fog network.

(c) Sample standard deviation of the average values of control messages for each new SON that joins the fog network (Fig. 3b).

Fig. 3: Analysis of the overhead for each new SON that joins the fog network.

in order to fill formerly created groups before creating new ones. This can be positive because during messaging, larger groups result in larger spanning trees which enhance the effect of the optimization logic.

According to the process that new SONs are added, $S_{far}$ re-joins the fog network through the $S_{new}$ that took its place. However, $S_{new}$ belongs to only one group (since it just joined) to which $S_{far}$ is the most distant. Having $S_{far}$ re-join through $S_{new}$ results in the creation of a new group that contains only $S_{new}$ and $S_{far}$. Instead, it is better to have $S_{new}$ join $S_{far}$ because there is the chance that $S_{far}$ belongs to other groups as well, which are not at capacity.

## IV. Evaluation

For evaluating the SONs, we develop a prototype which implements the necessary functionality to build and maintain fog networks. The prototype implementation along with the code of the evaluation can be found online [29]. With these tools, we conduct an evaluation that focuses on network performance using the following process. First, we build an underlying network that resembles the Internet topology using PeerSim [30] and then, we use the code of the prototype in a simulation that builds a fog network on top of the generated topology. For this evaluation, we run 100 experiments which are performed on randomly generated underlying topologies in order to show results from various cases. These results are related to *i*) scalability (Section IV-B), *ii*) fault tolerance (Section IV-C) and *iii*) intelligent messaging (Section IV-D). However, before we discuss the results, we mention details and limitations of the developed prototype.

### A. Prototype Details and Limitations

The prototype is based on a Java Web server developed using Spring Boot 2.0 and JgraphT 1.2.0 (a library for creating and processing graphs). By using JgraphT, every SON implements an object for each group graph, which stores all the necessary information about the neighbors. The reason we use JgraphT is that it includes the implementation of various graph processing algorithms (e.g., shortest path, spanning tree, etc.) which can be applied to the group graph objects. Such algorithms are useful for implementing some of the features of the SONs like intelligent messaging or joining based on shortest proximity.

The current version of the prototype aims at building a fog network for measuring network-related metrics. For this reason, a basic API is implemented along with the respective mechanisms to offer the following functionalities: New SONs organize each other in groups, as described in Section III-C. SONs also implement the fault tolerance mechanism in order to maintain connectivity in case of failures, as discussed in Section III-E. For communication within the fog network, intelligent messaging based on bandwidth minimization is implemented along with the querying mechanism, as discussed in Section III-F. Additionally, the improvement tweaks are implemented, as presented in Section III-I.

However, the system calls required for executing applications described in Section III-D, are not completed. Moreover, we do not consider resource constrained devices, as described in Section III-G. Notably, these functionalities are not necessary for this evaluation since our goal is to show that: *i*) the self-organizing mechanism for adding new compute nodes to a fog network, does not generate considerable overhead, *ii*) fog networks that follow a flat model (rather than the hierarchical) can cope well with failure and *iii*) intelligent messaging provides significant benefits compared to direct messaging.

### B. Scalability

To evaluate the scalability of the proposed approach, we analyze the overhead of adding new SONs from the cloud and the edge, to a fog network. Starting from a fog network of one SON and by adding another one until the network consists of 10,000 SONs, we measure the number of control messages exchanged for each new SON that is added (Fig. 3). This metric also includes the necessary messages to keep the local data stores updated. In Fig. 3a, we plot the *average* results (from 100 randomly generated scenarios) for different values of maximum group size. The values we choose correspond to small groups (5 members), medium groups (10 members), large groups (20 members) and very large groups (40 members). Fig. 3b plots again the average values of using very

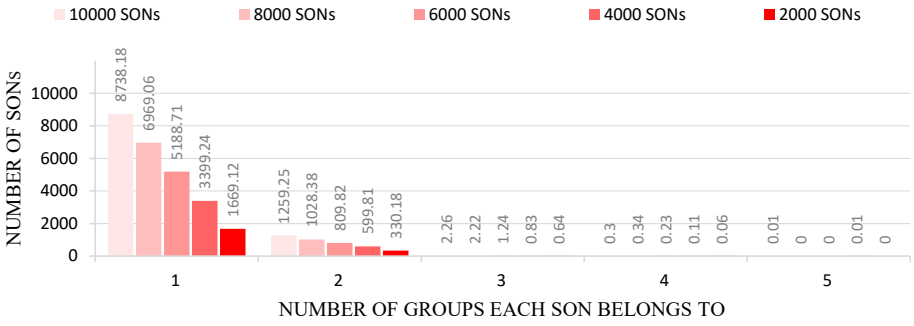



Fig. 4: Analysis of the number of groups that each SON belongs to in various fog network sizes.

large group sizes. However, to provide a better approximation of the general case, Fig. 3b also plots the *minimum* and *maximum* values (among 100 random scenarios) excluding 5% of extreme minimum and maximum values, respectively. The sample standard deviation of the average values (of Fig. 3b) is shown in Fig. 3c. The other group sizes exhibit similar behavior. To acquire these results, contact SONs are selected randomly based on a uniform distribution.

Moreover, Fig. 3b shows the regression lines formed by the Least Squares method that aim at depicting the dependence of the overhead on the size of the fog network. The regression equations are: $y = 0.0011x + 442.2$ with coefficient of determination $R^2 = 0.0106$, $y = 0.0005x + 199.55$ with coefficient of determination $R^2 = 0.0038$ and $y = -0.00001x + 65.18914$ with coefficient of determination $R^2 = 0.00001$ for maximum, average and minimum, respectively. The significance $F$ of the regression outputs for the average and the maximum are less than .0001. For the minimum, $F$ is .7476.

Fig. 3a shows that the average number of control messages stabilizes without intense irregularities and does not necessarily increase for each new SON independently of the group size. The regression results of Fig. 3b, also support this observation. Specifically, the regression output of the *average* case shows a significance $F$ value of less than 0.0001 which means that we can reject the null hypothesis that the variables are unrelated. The coefficient of determination $R^2$ is very low (0.0038), indicating that the average protocol overhead is not explained by the size of the fog network. The regression coefficient is also very low (0.0005) and suggests that for each unit of increase in the network size, the overhead increases by 0.0005 which is a very slow rate of increasing overhead. The *maximum* values exhibit similar behavior with a significance $F$ value of less than 0.0001, low coefficient of determination (0.0106) and a regression coefficient that suggests 0.0011 increase of the overhead for each unit of increase in the fog network size. The coefficient of determination for the *minimum* case is also very low (0.00001) and the regression coefficient suggests 0.00001 overhead reduction for each unit of increase in the network size. The significance $F$ value is 0.7476 meaning that we do not reject the null hypothesis that the variables are unrelated. Therefore, based on these results we can claim that the overhead of new SONs being added to a fog network does not depend on the network size.

TABLE II: Standard deviation of the average values in Fig. 4.

| | 2000 SONs | 4000 SONs | 6000 SONs | 8000 SONs | 10000 SONs |
|---|---|---|---|---|---|
| 1 group | 22.50 | 36.15 | 40.13 | 57.79 | 65.58 |
| 2 groups | 22.49 | 36.25 | 40.08 | 57.69 | 65.52 |
| 3 groups | 0.84 | 0.96 | 1.10 | 1.61 | 1.64 |
| 4 groups | 0.23 | 0.31 | 0.56 | 0.66 | 0.57 |
| 5 groups | 0.00 | 0.10 | 0.00 | 0.00 | 0.10 |

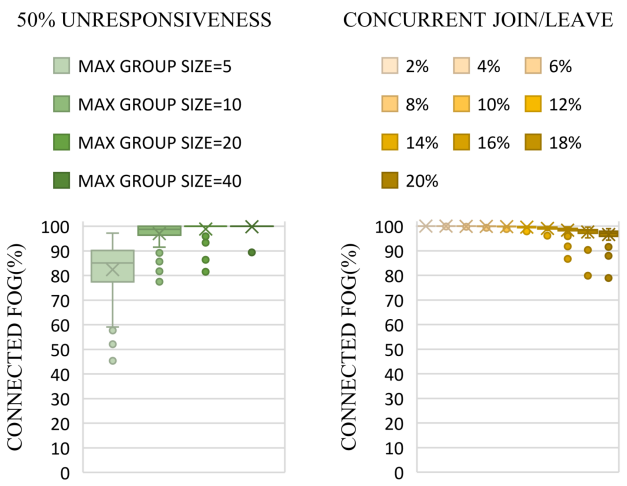
Since we do not use the max number of groups, as discussed in Section III-G, we present Fig. 4 which shows the average number of groups that the SONs of the fog network (various sizes) belong to, and Table II which shows the respective sample standard deviations. This experiment is performed in very large groups. Nevertheless, the other group sizes exhibit similar behavior.

Interestingly, we note that the vast majority of SONs belongs to one group and the number of SONs that belong to more groups decreases rapidly when the number of groups increases. The reason that this metric is important is that it shows that while the fog network grows, SONs do not impose on each other to store a growing amount of information.

The results of these experiments show that the overhead of adding new SONs does not depend on the fog network size which means that when the size of a fog network grows, the overhead does not increase. This is an objective of the SONs, as discussed in Section III-H, which advocates scalability. According to a recent literature review [31], fog systems need to operate at large scale, while most of the examined approaches do not meet the scalability criterion. Therefore, showing that fog networks scale well when the compute nodes follow the flat model using self-organizing mechanisms, may have a positive impact on the design and implementation of fog computing systems.

### C. Fault Tolerance

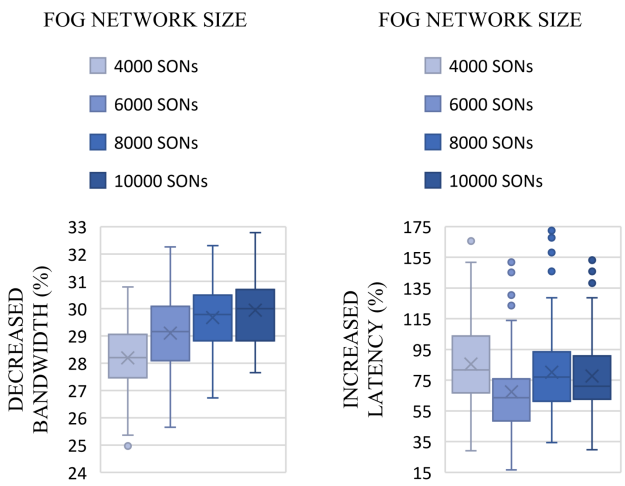
To analyze how efficiently SONs cope with failure, we present Fig. 5 (with results from 100 randomly generated sce-




(a) Percentage of responsive SONs that remain connected after 50% of the fog network has become unresponsive.

(b) Percentage of responsive SONs that remain connected after various unresponsiveness rates are induced.

Fig. 5: Analysis of fault tolerance.




(a) Decreased bandwidth utilization by using optimization logic when messaging within fog networks.

(b) Increased latency by using optimization logic that minimizes bandwidth when messaging within fog networks.

Fig. 6: Analysis of intelligent messaging.

narios). First, we show how well the fault tolerance mechanism discussed in Section III-E, is able to deal with unresponsive SONs. To do so, we use group sizes of 5, 10, 20 and 40 SONs, and we configure randomly 50% of the SONs to be unresponsive. Consequently, we initiate a global query which refines the fog network. Then, we measure the percentage of responsive SONs that remains connected and plot it in Fig. 5a.

Moreover, to assess fault tolerance under dynamic conditions with SONs that join/leave concurrently, we design the following process. We configure randomly 2% of the SONs to be unresponsive, initiate a global query to refine the fog network and measure the percentage of the responsive SONs that remain connected. After that, we have all the formerly unresponsive SONs join the fog network again and we add 2% to the unresponsiveness rate. This process continues repeatedly until unresponsiveness reaches 20%. Fig. 5b plots the results.

Fig. 5a shows how the max group size affects the fault tolerance of the fog network and how larger groups are more resilient to failures. Additionally, Fig. 5b shows results from assessing fog networks under dynamic conditions. Even though device failure is likely to occur in fog computing, current studies do not address fault tolerance [7], [8]. Therefore, results which show that fog computing using self-organizing mechanisms can cope well with failures, provide a promising research direction.

### D. Intelligent Messaging

To evaluate the effect of applying intelligent messaging (i.e., integrating a proximity measure and respective optimization logic) as discussed in Section III-F, we implement the following process. First, we build a fog network using the SONs. Then, we build another distributed network (used as a baseline) based on the same logic of using groups but without placing new compute nodes in the group of the closest proximity. In the former case, messages are sent using bandwidth minimization whereas, in the latter case each message is sent directly to all the neighbors of each compute node. In both cases, we initiate a global query from one compute node to the whole network. Consequently, we measure bandwidth utilization as expressed in the cost function of the optimization problem (cf. Section III-F), i.e., the number of hops needed to be traveled until all nodes receive the query. Fig. 6a shows the percentages by which bandwidth utilization is reduced when applying intelligent messaging, for various fog network sizes.

Notably, the messaging strategy we use favors bandwidth utilization but disregards latency delays. In the baseline, when messages are sent to all compute nodes directly, the latency might be lower. For this reason, we repeat the previous experiment and measure latency as the number of hops in the path of the query that is received by the last compute node. Fig. 6b shows the latency penalty due to the applied cost function, for various fog network sizes. These values represent the percentages by which latency is increased compared to the baseline. All values of Fig. 6 are again based on the average of 100 randomly generated scenarios.

We consider these results satisfactory because we use a bandwidth-related cost function. Fig. 6a shows a bandwidth gain of around 30% with a slight tendency to improve while the size of the fog network grows. The advantage of this approach is that the proximity measure and the cost function can be modified to pursue various goals in terms of performance, as discussed in Section III-F.

## V. Conclusion and Future Work

Within this paper, we propose self-organizing compute nodes which build fog networks without following the com-

monly applied hierarchical approach. SONs implement mechanisms for organizing themselves based on proximity, for achieving fault tolerance and for applying intelligent messaging based on optimization logic. From the experiments we conduct using SONs, we deduce that fog computing can benefit from self-organizing mechanisms due to the following: *i*) The overhead of the control messages when adding new compute nodes, remains stable despite the growing size of the fog network, which advocates scalability. *ii*) Fog networks become resilient and maintain connectivity even when participating compute nodes fail or disconnect from the network, which makes fog computing fault tolerant. *iii*) Compute nodes can communicate using intelligent messaging based on a configurable cost function which enables fog computing to pursue various goals in terms of performance.

Despite the promising results, we consider this work primarily as a starting point for further research in the field of flat models for fog networks. So far, our focus has been on communication aspects while future directions in this field include deploying fog computing applications on self-organizing compute nodes in order to evaluate application execution metrics. Regarding the arrangement of the compute nodes, future work can explore techniques for shifting participating nodes to groups with nodes of closer proximity at runtime, which may improve the disposition of the fog network over time. Moreover, designing mechanisms for ensuring system stability while integrating sensor networks in fog computing scenarios is also a promising research direction.

## REFERENCES


[1] 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.

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

[3] 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.

[4] W. Li, I. Santos, F. C. Delicato, P. F. Pires, L. Pirmez, W. Wei, H. Song, A. Zomaya, and S. Khan, "System modelling and performance evaluation of a three-tier cloud of things," *Future Generation Computer Systems*, vol. 70, pp. 104–125, 2017.

[5] V. Karagiannis, "Compute node communication in the fog: Survey and research challenges," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 1–5, ACM, 2019.

[6] V. Karagiannis, A. Venito, R. Coelho, M. Borkowski, and G. Fohler, "Edge computing with peer to peer interactions: Use cases and impact," in *Workshop on Fog Computing and the IoT (IoT-Fog)*, pp. 1–5, ACM, 2019.

[7] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[8] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[9] J. C. A. Leitão and L. E. T. Rodrigues, "Overnesia: A resilient overlay network for virtual super-peers," in *33rd International Symposium on Reliable Distributed Systems (SRDS)*, pp. 281–290, IEEE, 2014.

[10] D. Poola, M. A. Salehi, K. Ramamohanarao, and R. Buyya, "A taxonomy and survey of fault-tolerant workflow management systems in cloud and distributed computing environments," in *Software Architecture for Big Data and the Cloud*, pp. 285–320, Elsevier, 2017.

[11] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G.-J. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined fog-to-cloud (f2c) architectures," *Future Generation Computer Systems*, vol. 87, pp. 1–15, 2018.

[12] P. Bellavista, A. Zanni, and M. Solimando, "A migration-enhanced edge computing support for mobile devices in hostile environments," in *13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pp. 957–962, IEEE, 2017.

[13] S. Deng, Z. Xiang, J. Yin, J. Taheri, and A. Y. Zomaya, "Composition-driven iot service provisioning in distributed edges," *IEEE Access*, vol. 6, pp. 54258–54269, 2018.

[14] C. Prazeres and M. Serrano, "Soft-IoT: Self-organizing FOG of Things," in *30th International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 803–808, IEEE, 2016.

[15] S. Aditya and R. J. Figueiredo, "Frugal: Building degree-constrained overlay topology from social graphs," in *1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 11–20, IEEE, 2017.

[16] G. Tato, M. Bertier, and C. Tedeschi, "Koala: Towards lazy and locality-aware overlays for decentralized clouds," in *2nd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2018.

[17] M. Ali, N. Riaz, M. I. Ashraf, S. Qaisar, and M. Naeem, "Joint cloudlet selection and latency minimization in fog networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4055–4063, 2018.

[18] X. Song, Y. Huang, Q. Zhou, F. Ye, Y. Yang, and X. Li, "Content centric peer data sharing in pervasive edge computing environments," in *37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 287–297, IEEE, 2017.

[19] B. Yang, W. K. Chai, Z. Xu, K. V. Katsaros, and G. Pavlou, "Cost-efficient nfv-enabled mobile edge-cloud for low latency mobile applications," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 475–488, 2018.

[20] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[21] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in *1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 115–124, IEEE, 2017.

[22] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*, pp. 103–130, Springer, 2018.

[23] Y. Liu, J. E. Fieldsend, and G. Min, "A framework of fog computing: Architecture, challenges, and optimization," *IEEE Access*, vol. 5, pp. 25445–25454, 2017.

[24] V. Karagiannis and A. Papageorgiou, "Network-integrated edge computing orchestrator for application placement," in *13th International Conference on Network and Service Management (CNSM)*, pp. 1–5, IEEE, 2017.

[25] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *1st International Conference on Fog and Edge Computing (ICFEC)*, pp. 105–114, IEEE, 2017.

[26] P. C. Pop, W. Kern, and G. J. Still, "The generalized minimum spanning tree problem," Faculty of Mathematical Sciences, University of Twente, 2000.

[27] S. Brennan and M. Rabinovich, "Improving communication through overlay detours: Pipe dream or actionable insight?," in *38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1422–1431, IEEE, 2018.

[28] M. A. Kaafar, T. Turletti, and W. Dabbous, "A locating-first approach for scalable overlay multicast," in *25th International Conference on Computer Communications (INFOCOM)*, pp. 1–2, IEEE, 2006.

[29] "Son project repository," in *www.bitbucket.org/BasilKaragiannis/sonproject/*. Accessed online: 10 Jan. 2019.

[30] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *9th International Conference on Peer-to-Peer Computing (P2P)*, pp. 99–100, IEEE, 2009.

[31] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 416–464, 2017.