Sunstone: Navigating the Way Through the Fog

Julien Gedeon, Sebastian Zengerle, Sebastian Alles, Florian Brandherm, Max Mühlhäuser
Telecooperation Lab, Technische Universität Darmstadt, Germany
Email: {gedeon, zengerle, alles, brandherm, max}@tk.tu-darmstadt.de

***Abstract*—Fog Computing extends Cloud Computing by placing resources in the core network between cloud resources and (mobile) users. While a lot of frameworks to implement Fog Computing have been proposed, most do not consider loose coupling between federated fog resources. In such environments, the scalable discovery and orchestration of fog resources across multiple administrative domains and autonomous systems remains a challenge.**

In this paper, we present *Sunstone*[1], a discovery mechanism for Fog Computing that works on Internet-scale by combining on-path and off-path discovery mechanisms. Leveraging protocols present in the global routing infrastructure, Sunstone operates across multiple fog-domains while requiring no modifications to existing network middleboxes in the transit network. Furthermore, Sunstone includes a customizable orchestrator, which—given the results of the discovery process and application-specific policies—allows for a QoS-aware placement of applications in the fog.

***Index Terms*—fog computing, edge computing, mobile edge computing, discovery, orchestration**

I. Introduction

The advent of applications that generate large streams of data or for which processing latency is critical has led to the emergence of *Fog Computing* [1], [2] and—denoting a similar concept—*Edge Computing* [3], [4]. Example use cases include the processing of IoT data [5], Augmented Reality [6], Virtual Reality [7] and connected cars [8]. Fog Computing leverages resources in close proximity to the end user for computation offloading and, therefore, this new paradigm bridges the gap between local, customer-owned networks and distant Cloud Computing infrastructures. *Fog sites* provide computing and storage resources, and support multi-tenancy and isolation via lightweight virtualization, e.g., through containers [9]. Besides lower end-to-end latency, Fog Computing can also reduce the bandwidth utilization in core and transit networks.

Following this trend, several frameworks and architectures for Fog Computing have been proposed [10]–[13]. The IEEE recently defined a standard to adopt the *OpenFog reference architecture for fog computing* [14]. The reference architecture describes an approach with a hierarchical fog deployment model. This model allows the combination of fog and cloud deployments by defining the interactions between them. Moreover, it defines the usage of multiple fogs in the same deployment model. In such a fog architecture, two problems need to be addressed: (i) the discovery of fog resources, and (ii) the orchestration of those resources. To leverage resources between the cloud and the user, a multitude of fog sites in different autonomous systems (ASes) on the path from the user to the cloud should be taken into account. These networks are owned by various stakeholders, such as ISPs, other companies, or private customers. This leads to a loosely coupled $n : m : o$ relationship between users, fog resources, and cloud resources. Enabling a loose coupling between users and fogs is crucial to enable new business opportunities, in which different stakeholders (e.g., ISPs, network operators, cloud providers, hardware manufacturers) can offer computing resources. Federation between the different users and resources across administrative domains requires the definition of several interfaces. Figure 1 shows this general architecture of federated fog sites and the different interfaces for interaction between the cloud, fogs, and users.

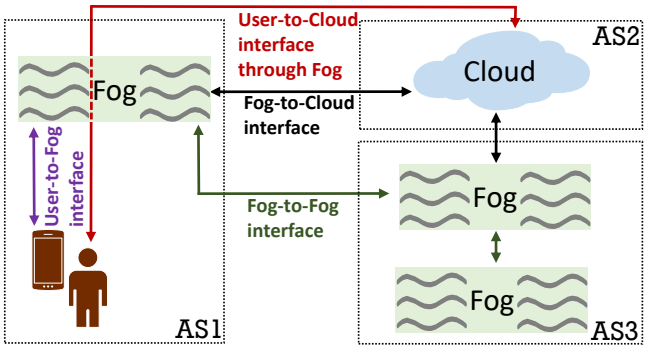


Figure 1. Reference Architecture of Federated Fogs in Different Autonomous Systems

This reference architecture does not match today's reality, where we mostly see single-cluster solutions [11] or centralized approaches [10], [15], [16] that are tightly coupled in the sense that they expect a direct relationship between the service running in the fog and the user end device. In addition, most existing frameworks focus mainly on the orchestration part. Regarding discovery, some assume that all available fog sites are known a priori and do not change [10]–[12]. However, due to the global scale, it is impossible to provision all parties with information about all fog sites. Furthermore, Fog Computing is inherently dynamic in the sense that fog sites may join and leave the network opportunistically. Other approaches implement discovery mechanisms that are based on broadcast [17], multicast [18] or centralized databases [19]. Those are either not scalable or—in the case of multicast—incompatible with today's global routing infrastructure. Peer-to-peer approaches, on the other hand, suffer from topology

[1]According to the icelandic allegory *Rauðúlfs þáttr*, a device called *Sunstone* was used in medieval times as a navigation aid in overcast skies.

mismatch, i.e., they create overlay topologies that do not correlate with the physical network topology. This jeopardizes the potential benefits of Fog Computing because no sensible placement decisions can be made for applications that have stringent constraints on the quality of service (QoS), e.g., with regards to latency. In conclusion, today no discovery mechanisms exist, which are globally scalable, support loose coupling, and consider the physical topology of the network.

In this paper, we present *Sunstone*, a novel approach for the joint discovery and orchestration of Fog Computing resources. Sunstone finds resources in the fog between users and clouds by combining three different discovery mechanisms, using both on-path and off-path discovery. It does so while being compatible with existing middleboxes and leveraging only protocols used in the global Internet (e.g., DNS and BGP). Hence, fog sites on the entire spectrum between cloud and users and residing in different autonomous systems can be discovered. Furthermore, Sunstone is globally scalable and supports loose coupling to dynamically leverage in-network fog resources. As a second contribution, Sunstone includes an orchestration component that manages the topology of deployed resources according to customizable policies. To the best of our knowledge, this is the first discovery and orchestration mechanism for fog resources that is globally scalable, supports $n : m : o$ relationships between users, clouds and fogs, and jointly uses on-path and off-path discovery mechanisms while considering the physical topology of the network to ensure QoS.

In summary, this paper makes the following contributions:

- We propose a scalable discovery mechanism for fog resources across multiple administrative domains by combining on-path and off-path approaches.
- We embed our discovery mechanism in an end-to-end orchestrator for the deployment of services in the user-fog-cloud continuum. The orchestrator provides a framework to define custom policies that are used for the placement decision of services following the results of the discovery process.
- We demonstrate the viability of our approach in a global network environment. We conduct a large-scale empirical study to report on the ability of Sunstone to find fog sites and quantify the overall benefit in terms of reduced end-to-end latency.

The remainder of this paper is structured as follows. We provide background information and review related work in Section II. Section III presents our approach with implementation details given in Section IV. We report and discuss our results in Section V and conclude the paper in Section VI.

II. Background and Related Work

A. Mobile Cloud Computing, Fog Computing, and Edge Computing

Offloading computations is especially beneficial for mobile devices because of their limitations with regards to available resources and battery life [20], [21]. Traditionally, this has been done via *Mobile Cloud Computing* (MCC), i.e., by leveraging public or private cloud resources [22]. However, emerging classes of applications require ultra-low latency and generate vast amounts of data—both of which prohibit the usage of cloud resources. For these reasons, the concepts of Fog Computing [1], [2] and Edge Computing [3], [4] have emerged. These paradigms aim to place resources close to (mobile) clients and hence, reduce the end-to-end latency and bandwidth stress in core networks. An extensive survey, outlining both concepts of Fog and Edge as well as their potential applications can be found in [3]. For the remainder of this paper, we assume that *Fog Computing* covers all available computing resources between user end devices and cloud infrastructures. Closely related is the concept of cloudlets [23]—small-scale data centers that offer proximate computing resources.

A number of frameworks have been proposed to offload computations to nearby resources, many of which are targeted at specific classes of applications. For example, Dupont et al. [12] present a platform for the migration of IoT functions across the cloud, edge and IoT gateways. Golkarifard et al. [24] develop an offloading system for wearable computing. Others focus on the specific access networks, inside of which functions are offloaded, e.g. cellular [13] or WiFi [25]. Our discovery and orchestration approach takes a holistic view and is not restricted to a specific offloading framework.

B. Discovery

The discovery of resources and available devices has been well-studied in the field of web services [26] and the Internet of Things (IoT) [27]. For web services, some approaches use broadcast and multicast, e.g., WS-Discovery[2] and Jini/Apache River[3]. Since these solutions have similar scalability requirements to fog discovery, different approaches were developed which are using centralized repositories. One example is UDDI [19], which does discovery by means of a central database. Federation is implemented on the client-side, who have to search in all registries. This approach is very forward-looking but still lacks the ability to have complex queries related to the topology (e.g., in terms of expected latency or bandwidth), which are required to find the best-suited fog node. UDDI merely provides simple queries on individual fields (e.g., location, business, or provider).

Cheng et al. [28] propose a framework for the programming of IoT services. Based on the NGSI standard[4], their approach relies on a centralized repository for discovery, to which services have to register. Klauck et al. [29] and Antonini [30] leverage multicast DNS. Some decentralized approaches exist [31], [32], but they focus mostly on the definition of information models. Moreover, in contrast to our approach, they neither leverage DNS as a globally scalable and decentralized database nor BGP as an information-exchange protocol to advertise fog sites as part of prefixes in the Internet. Instead, those approaches create their own hierarchical repositories.

[2]http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html (accessed: 2020-01-26)

[3]https://river.apache.org/ (accessed: 2020-01-26)

[4]http://www.openmobilealliance.org/release/NGSI/ (accessed: 2020-01-26)

The context for resource discovery in Fog Computing is totally different from that of web services and IoT. One large difference is that an intelligent discovery method has to work on a global scale. This prevents the usage of protocols that use broadcast (e.g., Multicast DNS) and the native discovery methods of wireless communication protocols, e.g., Bluetooth SDP [17]. Daza et. al propose hierarchical blockchains for IoT discovery [33], which rely on broadcast messages and therefore have limited scalability.

Some existing discovery approaches are targeted at Fog Computing. Closest to our work is the work of Zavodovski et al. [34]. Their discovery mechanism uses the SRV record of DNS to indicate locations of edge servers. In the first phase, the SRV records are retrieved from every domain returned by a traceroute. In the second phase, the services are onloaded to discovered edge servers. However, no details about the placement and orchestration mechanisms are provided. Using the SRV resource records is also less powerful compared to the NAPTR records we use. Furthermore, the presented approach relies on a centralized inter-domain orchestration and only supports on-path DNS zones. Similarly, *AirBox* by Bhardway et al. [15] and *MIDAS* by Abujoda et al. [16] also rely on a centralized repository.

Another approach to discovery is the use of overlays, as generated by peer-to-peer (P2P) and publish/subscribe protocols. For example, Gedeon et al. [35] propose to distribute available surrogates for Edge Computing on wireless home routers using Distributed Hash Tables (DHTs). DHTs are also used for P2P routing in information-centric networks (ICN) [36]. This approach has limitations regarding the discovery of the first hop, which has to be configured manually. More importantly, overlay networks typically do not correlate with the underlying physical topology. Hence, the overlay network cannot represent properties of the underlay, e.g., the latency on network links, which are crucial to make sensible placement decisions for fog applications.

C. Orchestration

Fog Computing to a great extent relies on container-based virtualization to achieve both isolation and multi-tenancy of application components [37]. *Kubernetes*[5] has emerged as the predominant orchestration tool for containers as of today. While Kubernetes recently was extended to support the management of multiple clusters[6], it still falls short in enabling a loosely coupled federation of resources between multiple clusters or even the dynamic discovery of unknown clusters. Therefore this solution does not scale well. Furthermore, Kubernetes does not take into account the location of the users for the placement decision.

EdgeNet [10] is a globally distributed Kubernetes cluster running on a large number of nodes spread over multiple continents. EdgeNet is physically distributed but logically represented by a single cluster running in a single administrative domain. Each participant is considered as a namespace part of the entire cluster and runs containers on multiple nodes. Since a single cluster under one administrative domain is used, the solution contradicts the actual architecture of the Internet, which is based on distributed autonomous systems and split administrative responsibilities. *KubeEdge* [11] is another example of a single-cluster solution. In summary, both EdgeNet and KubeEdge rely on centralized controllers, while the compute nodes are physically distributed. Another proposed approach is to use a special proxy inside Kubernetes, which mitigates the shortcoming of missing proximity-aware traffic routing [38]. This solution also relies on a central controller. In contrast, Sunstone is a framework for distributed and loosely coupled discovery and orchestration of resources over multiple administrative domains.

[5]https://kubernetes.io/ (accessed: 2020-01-11)

[6]https://github.com/kubernetes-sigs/kubefed (accessed: 2020-01-22)

III. Sunstone: Scalable Discovery and Orchestration for Fog Computing

In this section, we describe the main contributions of Sunstone. A distributed discovery agent uses a combination of three approaches to find available fog resources. The result of the discovery is the basis for the subsequent orchestration that ultimately determines where applications are run. To do so, the applications' placement policies are mapped to the properties of the discovered resources. This high-level system overview of Sunstone is shown in Figure 2.

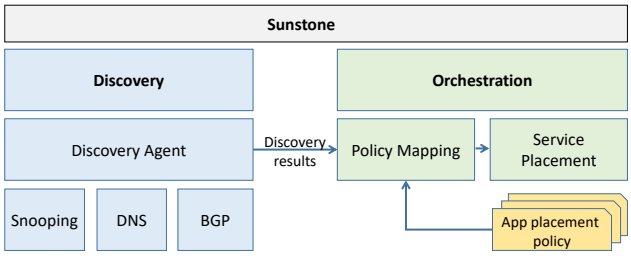


Figure 2. Overview of Sunstone

A. Discovery

For our discovery mechanisms, we use both on-path and off-path approaches. We start from the assumption that an application requested by a user runs in the cloud. In search of a suitable fog site, both the cloud and the users can be the initiator of the discovery procedure by issuing an API call to the discovery component of Sunstone. The actual discovery is carried out by a discovery worker process running in the cloud where the user's application is located.

On-path discovery: On-path discovery finds fog sites located on the actual network path taken between users and their cloud applications (Figure 3(a)). Hence, these fog sites are located on network middleboxes that are involved in the forwarding and routing process of the application's payload between the user and cloud. This approach reflects the deployment model in Cisco's initial vision of Fog Computing [2], where networking hardware (e.g., routers) make extra resources available to carry out computations. Another example is customer-premises

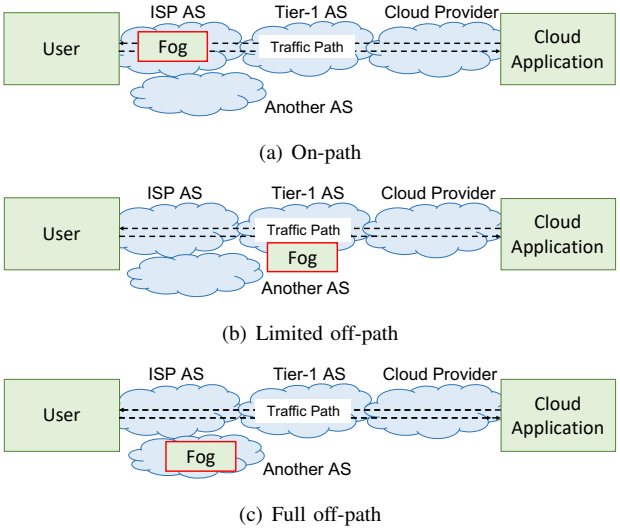


Figure 3. Discovery Approaches

equipment (CPE), e.g., one's home gateway. On-path discovery is done by sending probe packets that are recognized and answered (*snooping*) by network middleboxes that host a fog site. Because most of this hardware is optimized for quick forwarding of incoming packets, we consider a special case to also be on-path, namely when the middlebox does not process the traceroute packets itself, but is able to duplicate and forward them to a snooper. As an example, routers can efficiently implement this using the 5-tuple rules[7].

Off-path discovery: In contrast, the scope of off-path discovery mechanisms is beyond this direct user-to-cloud path. We call mechanisms that restrict the discovery to fog sites residing within the autonomous systems on the direct path *limited off-path* (Figure 3(b)). Limited off-path discovery can, for example, find fog sites that are strategically placed in the backbone of an ISP. In *full off-path* discovery (Figure 3(c)), we can also discover fog sites that are located in autonomous systems not traversed on the direct path.

Both of these approaches have particular challenges. In on-path discovery, we first have to accurately construct the path that requests take. Intermediate network middleboxes on the path, such as ECMP load balancers or network address translators make this path non-trivial to determine. For off-path discovery, we need to build a network graph that traverses other autonomous systems, starting from the user-to-cloud path. This is challenging, given the global extent of the Internet. On one hand, to retain the potential benefits of Fog Computing, it might not seem sensible to extend the search for a fog site beyond the path length of the user's initial connection to the cloud. On the other hand, logical paths in autonomous systems do not necessarily correspond to the actual routing connections, as we will outline in Section V-D.

Sunstone jointly uses three discovery mechanisms, combining on-path with off-path discovery: (i) Snooping of traceroute packets, (ii) DNS NAPTR record information, and (iii) BGP community string advertisement. These three mechanisms are run subsequently and the results are aggregated at the discovery worker. Table I shows which mechanisms are used for which class of discovery approach. Snooping is used to find on-path fogs, while DNS is mainly used for limited off-path discovery. For full off-path discovery, BGP is used to discover ASes that contain a fog site, while DNS is used as an auxiliary mechanism to validate and enrich the discovery.

[7]source IP, destination IP, source port, destination port, transport protocol

Table I
MECHANISMS FOR ON-PATH AND OFF-PATH DISCOVERY

	Snooping	**DNS**	**BGP**
On-path	✓		
Limited off-path		✓	
Full off-path		(✓)	✓

1) Traceroute snooping: Our first mechanism discovers fog sites on the direct path between the user and the cloud by snooping packets that we send on this path. For this, we carry out a variant of a traceroute from the cloud to the user. Traceroute tools determine which paths packets take in a network for a given destination. By incrementally increasing the TTL field in IP packets, traceroute eventually finds all hops on this path. Contrary to other approaches [16], this approach does not require any modifications to network middleboxes. To realize the discovery, we introduce a custom payload in the traceroute packets. Snoopers on fog sites along the path that receive such a traceroute packet in turn reply with a message containing information about the fog site. We describe the detailed implementation of our traceroute and the messages in Section IV-A.

2) DNS NAPTR record information: For each intermediate hop returned by the traceroute, we trigger our second discovery mechanism that finds fogs that are off-path but on-AS path. We suggest to provision information about fog sites within an AS via the *Naming Authority Pointer* (NAPTR) resource record (RR) of the Domain Name System (DNS). The hierarchical approach of the DNS infrastructure provides enormous scalability, making it a suitable decentralized database for fog sites. Traditionally, NAPTR records are used to translate E.164 telephone numbers to SIP URIs. Another use case is to determine different services, e.g., SIP via TLS or SIP for a telephone number. The service field of the NAPTR record can be provisioned with different values in different RRs to provide multiple service entry points, depending on the users' service selection. It is also possible to rewrite records with regular expressions, making NAPTR more flexible compared to other types of resource records, e.g., SRV.

For the DNS-based discovery, we first translate all IP addresses from our raw traceroute results to the respective AS number by using raw BGP update messages. Using PTR records, the network is resolved to its name. Afterward, we use NAPTR records to enrich the network of the intermediate hop with information about fog sites. This means that if our

traceroute traverses a hop, we first aggregate the IP to its CIDR network. By performing a reverse lookup on PTR records, we translate the network CIDR-block to its name. NAPTR records are used to attach metadata such as API endpoints of fog sites, their type, distance and further information to this network. Listing 1 shows an example of a NAPTR record, containing the name of the resource, the Internet (`IN`) and NAPTR type, a priority value, a weight for load-balancing, the response type, and the used service (`sunstone-k8`). In our example, the response type is an SRV record to allow dynamic port numbers for different API endpoints. The service field is used to allow different types of services (e.g., a Kubernetes cluster).

```
daFogNetwork.tk.tu-darmstadt.de.
IN NAPTR 10 100 "S" "sunstone-k8" ""
darmstadtFog.tk.tu-darmstadt.de.
```

Listing 1: Example NAPTR Resource Record

3) BGP community string advertisement: Lastly, we use an extension of the Border Gateway Protocol protocol (BGP) to realize full off-path discovery. DNS-Discovery limits us to on-path discovery because only intermediate hops between user and cloud can be used to advertise fog sites. A naive approach could be taken by using DNS to crawl over all domain names of all hops. However, this clearly does not scale. Therefore we propose to use BGP as another method to announce off-path networks containing fog resources. BGP is the standard protocol to exchange routing information between ASes. BGP uses tables that contain CIDR-prefixes and the respective path through ASes one needs to traverse. Besides this information, another feature of BGP are *community strings* [39], which provide additional information about prefixes to BGP peers in the form `ASN:Value`, where `ASN` is the unique number of the autonomous system and `Value` is a numerical value that represents the community. The community string is used to provide the network with metadata about other peers. This is done by sending a BGP update message to all neighbors. This information can, for instance, be used to implement certain routing policies. We suggest using this community string to advertise the presence of a fog site in an AS via a specific *fog community string*. More specifically, we use *large communities* [40]—an extension to the standard that lowers the probability of collisions—to tag a CIDR-prefix if it contains a fog site. The prefix containing fog sites is announced by a BGP peer on the Internet. Since the community string which is used as a tag is transitive, all peers will forward the prefix with the tag and thus make the information about the available fog site globally known.

Community strings by themselves have no fixed semantic meaning, although certain best practices exist, e.g., certain numerical ranges represent specific types of information and so-called *well-known community strings* exist. For a practical deployment of our approach, we suggest that a specific *fog community string* be standardized as a well-known community. Community strings are further limited by the amount of information they can convey. Therefore, DNS is also part of this discovery mechanism. If a fog community string is announced for an AS, a DNS query as described before is issued. This is done to ensure two things: (i) it validates the presence of a fog site (in case the fog community string was used by a third party for another purpose) and (ii) it provides us with the necessary information to access the fog site, e.g., its API endpoint. During the discovery procedure, a list of all tagged networks (i.e., networks for which a fog site has been announced via a community string) is used to discover off-the-path fogs. How many ASes should be traversed is determined by a parameter k. The k-value is equal to the AS path length. This naturally forms a tradeoff. For low values of k, we might miss suitable fog sites, while large values of k result in potentially unnecessary overhead. Table II shows the scope of the discovery for selected values of k.

Table II
SCOPE OF FOG SITES DISCOVERY FOR DIFFERENT VALUES OF K

k	**Scope of discovery**
1	AS of the user
2	Neighboring ASes
$d(user, cloud)$	ASes not further away than the cloud
∞	Global Internet

B. Orchestration

The orchestration component manages the topology and lifecycle of fog applications, such as their creation, update, and deletion. Most importantly, it is responsible to make placement decisions, i.e., to determine where applications are run. Because a single application can consist of multiple components, those have to be interconnected (e.g., by making them addressable) using a topology management function. To orchestrate resources, our orchestrator takes two inputs: (i) the results of the discovery and (ii) an application descriptor. The application descriptor contains the application's components (e.g., containerized services) and specifies a placement policy.

The policies are the core of the decision-making process for the placement and contain constraints and optimization goals for the placement. For example, an application's policy could be to select the fog site with the lowest latency to the user. Another policy could be used to aggregate data at an optimal location, such as determining the average sensor measurement data. One such use case is sensors that gather environmental data. At certain locations, depending on the aggregation function and amount of data, different sensor readings should be aggregated inside the network at fog sites. Based on the location of all fog sites, the best aggregation points are determined by using a tree data structure. In this tree, all sensors are represented as leaves, while fog sites are inner nodes. The selection of suitable fog sites is done by traversing the tree and aggregating a defined set of sensors on each node, based on data attributes (e.g., the minimum and maximum of sensors per fog).

We can further distinguish policies by the (single- or multi-) tenancy of users and applications. Especially for multi-

application policies, one of the core tasks of our orchestration is to manage the topology that interconnects all application components. We will outline examples in Section IV-B.

In detail, the orchestration process for deploying applications on a fog site consists of the following steps:

1) Enrichment of discovery metadata: The different discovery mechanisms as described in Section III-A do not only find available fog sites but already collect latency data on the paths between the cloud and users. Since various policies exist, which differ in their decision metrics, more measurements must be run. Those are for example bandwidth measurements (e.g., for bandwidth-intensive applications) or simply counting the number of devices that traverse a single hop (e.g., for the aggregation of sensor data). This measurement data is loaded as metadata into the discovery result set.
2) Policy mapping: Each application can use its own policy, which describes how to place services across fogs. In the first step, our policy mapper filters unwanted results. These can be fog sites that are incompatible wrt. the required execution environment or not commercially viable. Afterward, measurements are executed to extend the metadata with user-experienced metrics. The type of measurement is determined by the used policy. As an example, for a latency-critical application, this step will add the minimum, maximum and mean latency for each discovery result.
3) Service placement: During the service placement step the decision is taken where to run which service. With the extended result from the policy mapper, each application component is planned to be placed at the best-suited fog site. Lastly, the planned orchestration result is executed. Sunstone supports different types of execution environments (e.g., Kubernetes or OpenStack) and therefore carries out the placement in two steps. The first step is technology-agnostic and abstracts implementation details, while the second step is specific to the actual execution environment.

Besides this placement functionality, the orchestrator exposes an API to manage the lifecycle of already running applications. One of the operations is to delete an application, either at all sites or just decommission some deployed fog sites. Another kind of operation is to update existing applications, for example in an aggregation use case to make the grouping broader or denser.

IV. Implementation Details

We realize the concepts described in the previous section in a reference implementation. A high-level overview of the system architecture is shown in Figure 4. The main components are realized as a collection of six microservices (depicted as rectangles). In addition, we use a message queue (*RabbitMQ*) for communication and a database (*etcd3*) to persist runtime data. In addition, BGP and DNS data is used as auxiliary information for the discovery. All microservices are written in Python 3 using *Flask* as a framework. Low-level code, such as

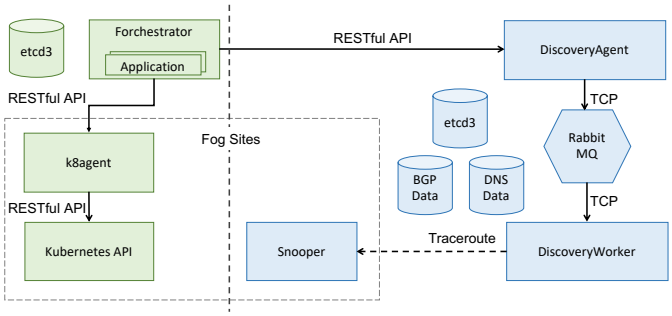


Figure 4. Implemented System Architecture

raw network operations, is written in C++ and integrated into Python using *Boost.Python*. Our system architecture consists of two functional domains. The first one is the discovery functionality (colored in light blue), which can also be used independently. The task of this function is to discover new fog sites. The second is the orchestration functionality (colored in light green), which is responsible for orchestrating the discovered resources according to the respective application policy. Furthermore, this part also carries out additional measurements that might be required by placement policies. It is important to note that applications can consist of multiple components that are to be placed on fog resources. The orchestrator requires a technology-dependent component to interact with the runtime environment at the particular fog site. Our reference environment for running components is Kubernetes. Therefore, a distributed service of the topology function (*k8agent*) is run on the fog site.

As can be seen from Figure 4, each of the two functionality domains is composed of multiple microservices. The microservices running at a fog site (denoted by a dotted rectangle) are fully decentralized. Each application needs a decision logic (e.g., composing the policy), which runs on a centralized controller. The application workload itself is distributed across different fog resources. Each cloud site can run a set of centralized controller components, which run multiple applications. Fog sites are shared between controllers, to allow for global scalability. Different applications (that also could be managed by different central controllers) can share fog resources (fog multi-tenancy). Controllers share the same pool of fog resources, while being logically isolated and not allowed to access the resources of other controllers or even different apps. The composition of multiple centralized components on the cloud sites (n clouds), using different fog sites (m fogs shared with up to n clouds) and providing services to o user leads to the $n : m : o$-relationship of those entities as described in our reference model (see Figure 1).

A. Discovery Function: Discovery Agent, Discovery Worker, and Snooper

The *discovery agent* is mainly used as RESTful API front end to access all discovery-related functions from other microservices. Etcd3 is used as a backend to persist requests and aggregate the results of the discovery process. The

actual discovery process is executed by the *discovery worker*. The distributed coordination of discovery jobs is done by means of RabbitMQ and Python-Celery while coordination of distributed locks and storage is the task of an etcd3 database. The core functionality of the discovery process is to invoke traceroutes and perform the subsequent discovery steps (see Section III-A). Traceroute is a mechanism to determine the actual paths that packets take in a network [41]. Basic traceroute implementations follow a naive approach to gather path information in small networks. They neglect the existence of NAT (Network Address Translation) and ECMP (Equal-Cost Multi-Path), which are very common in today's networks. In practice, this might lead to false-positive links [42]. For these reasons we choose to use *Dublin Traceroute*[8] for our implementation, a variant that is able to detect NATs and does not have problems with NAT middle-boxes not according to standards. Our snooper detects traceroute packets from the cloud to the user and does not provide any invocable API. The snooper evaluates the payload of traceroute packets and returns the fog data to the discovery agent's API, which is specified in the packet. The packets have a specific payload, which contains the API endpoint of the discovery agent, the API version and a unique ID to refer to a running discovery process. The payload is encoded using *ProtoBuf*. Since our discovery solution can be used for different use-cases and fog execution environments (e.g., Kubernetes or OpenStack), the response contains the type of the discovery fog, as well as the distance from the cloud to that fog site and the API endpoint to reach the discovered fog site.

B. Orchestration Function: Forchestrator and K8agent

The *forchestrator* (fog orchestrator) is responsible for the lifecycle management of all created resources during runtime at the fog sites. In cases where the application consists of multiple application components, the orchestrator is responsible to ensure the interworking between the components. This is done by injecting runtime variables into the workload, such as mutual API endpoints of each service.

For each user, the orchestrator executes a discovery process and evaluates the policy with the discovery result to plan placements. Placement decisions are made based on *policies*. Since different applications can use different metrics to define QoS, a flexible decision-making process based on such policies is required. Policies are implemented independent of a specific application, to allow their reuse across different applications. Applications specify the concrete policy they want to be applied for the placement in an *application descriptor*. A policy can invoke so-called measurements, which gather additional information, e.g., by measuring latencies or available bandwidth. Policies exist with different characteristics in terms of the tenancy of application components and users. As described in Section III-B, policies are abstract implementations of the decision logic rather than application-specific. According to Hong et. al. [43], policies can be classified as follows:

- SaSu (Single application component - Single user): One fog application used by a single user.
- SaMu (Single application component - Multi-user): One fog application shared between multiple users.
- MaSu (Multi-application component - Single user): Multiple applications or components used by a single user.
- MaMu (Multi-application component - Multi-user): Multiple applications or components shared between multiple users.

Policies can be distinguished in an abstract way by the types described above. The implementation of such a type fulfills a more concrete use case, while deriving the basic properties such as application and user cardinality. The naming of concrete policies is based on the use-case and the policy type. Two examples of policies are (i) *nearest-SaSu*, which for a single user finds the nearest place to run a single application component, and (ii) *aggregation-SaMu*, which places a single application at the best place to aggregate data. Placement plans generated by applying the policies are independent of the execution environment. Depending on the concrete execution environment at hand, placement decisions are then translated to API calls of the concrete execution environment. This is done in the final orchestration step, where the forchestrator creates tenants on top of each fog resource to provide isolation and multi-tenancy. In our implementation, the *K8agent* is used as a helper microservice located at the fog sites. This service acts as a gateway to Kubernetes at each fog site. It provides two kinds of functionalities: (i) managing the tenants at the fog site and (ii) executing measurements, e.g., measuring the latency to users.

V. Evaluation

We evaluate Sunstone on Internet-scale. Our experimental setup is described in Section V-A. We then show how the discovery and orchestration components of Sunstone reduce the latency for users (Sections V-B and V-C). We further discuss our findings and their implications in Section V-D.

A. Experimental Setup and Methodology

We conduct an experimental study on top of the real network topology of the Internet's ASes. To do so, we use data from RIPEstat[9]. More specifically, we leverage measurement data collected from RIPE atlas[10] probes. These probes, deployed and operated by volunteers, provide connectivity and reachability information and can be used for custom measurements (e.g., to measure latencies). We start from a complete list of all ASes, retrieved from CAIDA[11] and enriched with metadata from RIPE. For simplicity reasons, we only consider ASes located in Germany for our evaluation. Furthermore, only ASes which contain at least one RIPE probe are used, which are reachable without NAT (important for the usage of ICMP in the traceroute and measurements), resulting in a total of 107 ASes. It is important to filter out probes with NAT in front,

[8]https://dublin-traceroute.net (accessed: 2020-02-06)

[9]https://stat.ripe.net/ (accessed: 2020-01-21)

[10]https://atlas.ripe.net/ (accessed: 2020-01-21)

[11]https://www.caida.org (accessed: 2020-01-21)

to use ICMP with them directly. We assume we have one cloud location, represented by an AWS EC2 instance located in the Frankfurt region. Probe metadata, such as the probe's IP address and geolocation is added from RIPEstat. We then construct a graph of all ASes. To do so, we run the same traceroute mechanisms as implemented in Section IV-A to construct a path from our cloud location to all probes. All traceroute results are aggregated by IP to AS-level data, using a database of prefix-to-ASN relationships. Information from raw BGP updates, such as prefix-to-AS mappings is loaded from Routeviews RIB (Routing Information Base), creating a graph of all ASes. The topology of the Internet on AS level is loaded from CAIDA (*as-rel dataset*). This data allows us to add paths that were not taken by our traceroute packets. The result is a graph of all German ASes, whose edges represent the connection from the cloud instances to all relevant probes. Figure 5 summarizes the process of creating this graph. We use this graph to run an evaluation of our discovery approaches, which we use as the topology for our evaluation.

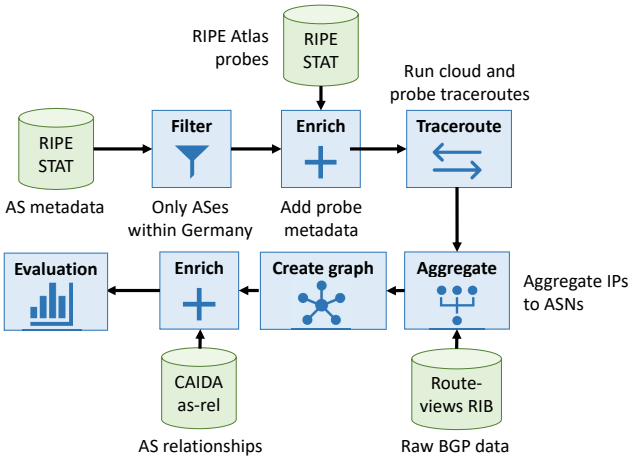


Figure 5. Evaluation Setup

B. Discovery

We first show how the combination of different discovery mechanisms reduces the resulting end-to-end latency from the user to the location where the service is executed. We simulate the discovery process as follows: We use the RIPE probes to represent the location of both users and fog sites. For the fog sites, we assume that either 5 % or 10 % of ASes contain at least one fog site. For each of those, we randomly choose one probe within that AS to represent this fog site. Note that for each AS, we were able to find a probe without NAT, meaning that we can measure the exact latency to the probe. The remaining probes are considered to be users. We start from the baseline where our service runs in the cloud on Amazon AWS infrastructure in the Frankfurt region and measure the hop count and latency via the same traceroute mechanism we use for discovery (see Section III-A1). Then, we perform discovery (i) via DNS only and (ii) including BGP data.

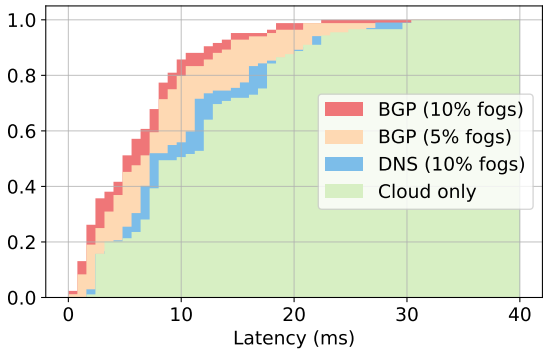


Figure 6. CDF of the End-to-End Latency Resulting from Different Discovery Mechanisms

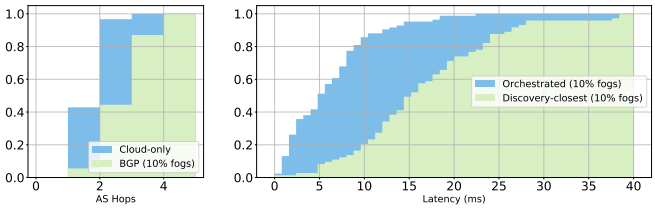
For DNS-based discovery, we first determine the shortest path from each user to our cloud instance by using our enriched graph. This graph contains a real traceroute from our cloud location to each user, which in most cases is the shortest path (physically shorter paths might exist but for instance, those might not be used because of economic reasons). On this path, we check for the existence of ASes which contain fog resources. For DNS, we assume that the shortest AS path is used and that each AS configures all NAPTR records correctly. If we include BGP-based discovery, we assume that the information about fog sites has been disseminated throughout the Internet. Since the BGP-method uses a list of all tagged networks, as described in Section III-A3, we use a list of all ASes with fog resources as well for the evaluation. We determine the k-value for each path from the users to each fog. For non-orchestrated results, we just use the nearest fog with the lowest k-value. For orchestrated results, we use RIPE Atlas probe data to determine the latency from each fog to each user. Afterward, we use only the best result for each user. The difference between orchestrated and discovery-closest is evaluated in Section V-C in more detail.

Figure 6 shows the CDF of the latency for DNS- and BGP-discovery, as well as the cloud baseline. The average latency from the cloud to the user is 13.32 ms, while it becomes marginally better if we use the DNS approach to find fog sites with 11.71 ms (-12.09 %). A great improvement is possible with the BGP approach, which already achieves an average latency of 8.05 ms (-39.56 %) with just 5 % of ASes containing fog sites. Doubling the amount of ASes with fog sites and using our BGP-based approach results in 6.98 ms (-47.6 %) latency on average. It is worth noticing, that BGP-discovery outperforms the DNS-only discovery, even with half the percentage of fog sites.

C. Orchestration

We now show how the previously obtained results improve if we include the orchestration component. Recall that if we use the discovery mechanism alone, the fog with the shortest paths to the user is chosen. Because this path length does not necessarily correlate with the latency of the path (e.g., when ISPs use AS prepending to increase the length artificially this

does not impact the latency), a suboptimal fog site might be chosen. In fact, when analyzing data depicted in Figure 7(a), we found that the AS path length to the best fog site in terms of latency could even be longer than the user to cloud path. To leverage this best fog site, we need a corresponding orchestration policy that is executed. Figure 7(b) compares the resulting end-to-end latency if we use only the closest fog from the discovery (in terms of hop count) versus a placement decision made by the orchestrator using the *nearest-SaSu* policy (see Section IV-B). Without orchestration, the average latency is even worse than using the cloud, because the most efficient fog is not selected (17.92 ms). When using the orchestrator we get a much better average latency (6.98 ms).



(a) CDF of the AS Hop Count

(b) CDF of the Latency Comparison

Figure 7. Evaluation of the Orchestration Component

D. Discussion and Future Work

We now discuss some of the implications and limitations of our proposed approach in a real-world environment:

Discovery time: In our evaluation, we have assumed that the discovery is triggered at the moment when the decision is made to migrate a (running) application to fog resources. We found that on average, a full discovery and orchestration, i.e., using all three mechanisms, carrying out measurements according to placement policies, and placing a container on Kubernetes, took between 14 s and 23 s. It is important to note that besides carrying out fog discovery on demand, we can also envision an asynchronous mechanism that periodically runs a discovery and caches the result.

Number of fog sites: In our evaluation setup, we have assumed that 5 % or 10 % of all ASes contain (at least) one fog site. We further motivate this choice by showing how this percentage affects the resulting possible latency. Figure 8 shows the results for percentages between 1 and 10. From the results, we can conclude that the gain in latency flattens out quickly and does not improve above 8 %. Note that the latency measured here is to the closest fog site and thus not necessarily the one that is found by the discovery.

Distribution of BGP communities: We have assumed that BGP community strings are not filtered. In practice, however, some of the ISPs will block all or some incoming communities (e.g., private communities) to prevent the triggering of control communities in their own network. Due to filtering at AS borders with increasing AS hop counts the probability that a community is filtered increases as well. Furthermore, there is a considerable delay, ranging from minutes to hours in the distribution of community strings. To mitigate this, one could employ *DNS Pre-Loading*. This method loads all PTR and NAPTR records from each prefix routed in the Internet beforehand, making all fogs within an AS known immediately. This method is very reliable but still has a delay in the availability of the information. In a measurement study, Streibelt et al. [44] report that more than 50 % of communities traverse more than four ASes. If we compare this finding with a RIPE study[12] that states the average AS hop count in IPv4 networks to be 4.3, we can conclude that the dissemination of community strings is sufficient in practice.

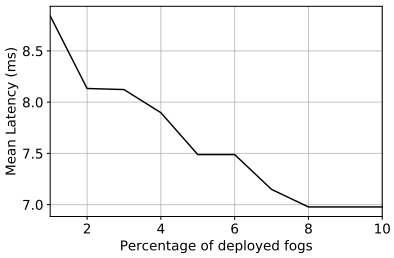


Figure 8. Average Latency for a Varying Number of Fog Sites

Scope of the discovery: For simplicity reasons, we have only considered ASes located in Germany to evaluate our discovery mechanisms. The results can easily be transferred to a larger scale, up to the global Internet. In fact, the demonstrated benefits (e.g., with respect to the improved end-to-end latency) in our restricted scope of ASes can be considered a worst-case scenario. If we would consider cloud instances that much more distant to the user (as is the case in many applications today), the benefits of our discovery and orchestration mechanisms would be even more striking. We leave this to explore for future work.

AS path asymmetry: The measurements carried out by the orchestrator do not take into account AS path asymmetry, i.e., measurements are conducted only on the path from the fog site or cloud to the user and not vice versa. In many cases, the chosen fog site will be close to the user, i.e., either in the same AS or in neighboring ASes. The closer the fog is to the user, the more likely we are to have symmetric paths, since path asymmetry mainly occurs if we traverse multiple ASes and hot-potato routing is used. Hence, we argue this one-way measurement is unlikely to have a negative impact in practice.

VI. Conclusion

In this paper, we presented Sunstone, a novel approach for the discovery and orchestration of federated Fog Computing resources. By combining three discovery mechanisms, Sunstone finds fog resources on a global scale, while remaining scalable. Following the discovery procedure, an orchestration component makes deployment decisions based on customizable application placement policies. We have demonstrated the viability of

[12]https://labs.ripe.net/Members/mirjam/update-on-as-path-lengths-over-time (accessed: 2020-01-28)

Sunstone in a real-world testbed, showing it results in lower end-to-end latencies and hence, improves the QoS of services.

ACKNOWLEDGEMENT

This work has been cofunded by the German Research Foundation (DFG) and the National Nature Science Foundation of China (NSFC) joint project under Grant No. 392046569 (DFG) and No. 61761136014 (NSFC), and as part of the Collaborative Research Center 1053 - MAKI (DFG). The authors would like to thank *man-da.de GmbH* for their support. Furthermore, this work was supported by the *AWS Cloud Credits for Research* program.

REFERENCES

[1] J. Gedeon, J. Heuschkel, L. Wang, and M. Mühlhäuser, "Fog computing: Current research and future challenges," in *1. GI/ITG KuVS Fachgespräche Fog Computing*, 2018, pp. 1–4.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and its Role in the Internet of Things," 2012, pp. 13–16.

[3] J. Gedeon, F. Brandherm, R. Egert, T. Grube, and M. Mühlhäuser, "What the Fog? Edge Computing Revisited: Promises, Applications and Future Challenges," *IEEE Access*, vol. 7, pp. 15 284–152 878, 2019.

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

[5] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *IEEE Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[6] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan, "Towards wearable cognitive assistance," in *Proc. MobiSys*. ACM, 2014, pp. 68–81.

[7] E. Cuervo, K. Chintalapudi, and M. Kotaru, "Creating the perfect illusion: What will it take to create life-like virtual reality headsets?" in *Proc. HotMobile*, 2018, pp. 7–12.

[8] R. Vilalta, S. Via, F. Mira, R. Casellas, R. Muñoz, J. Alonso-Zarate, A. Kousaridas, and M. Dillinger, "Control and management of a connected car using SDN/NFV, fog computing and YANG data models," in *Proc. NetSoft*, 2018, pp. 378–383.

[9] R. Morabito, V. Cozzolino, A. Y. Ding, N. Beijar, and J. Ott, "Consolidate IoT Edge Computing with Lightweight Virtualization," *IEEE Network*, vol. 32, no. 1, pp. 102–111, 2018.

[10] J. Cappos, M. Hemmings, R. McGeer, A. Rafetseder, and G. Ricart, "EdgeNet: A Global Cloud That Spreads by Local Action," in *Proc. IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 359–360.

[11] Y. Xiong, Y. Sun, L. Xing, and Y. Huang, "Extend cloud to edge with kubeedge," in *Proc.IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 373–377.

[12] C. Dupont, R. Giaffreda, and L. Capra, "Edge computing in IoT context: Horizontal and vertical Linux container migration," in *Proc. of Global Internet of Things Summit*, 2017, pp. 1–4.

[13] T. Taleb and A. Ksentini, "Follow Me cloud: Interworking federated clouds and distributed mobile networks," *IEEE Network*, vol. 27, no. 5, pp. 12–19, 2013.

[14] IEEE-1934-2018, "IEEE standard for adoption of openfog reference architecture for fog computing," IEEE Standard Association, Standard, 2018.

[15] K. Bhardwaj, M. Shih, P. Agarwal, A. Gavrilovska, T. Kim, and K. Schwan, "Fast, scalable and secure onloading of edge functions using airbox," in *Proc. IEEE/ACM Symposium on Edge Computing (SEC)*, 2016, pp. 14–27.

[16] A. Abujoda and P. Papadimitriou, "MIDAS: middlebox discovery and selection for on-path flow processing," in *Proc. COMSNETS*, 2015, pp. 1–8.

[17] S. Avancha, A. Joshi, and T. Finin, "Enhanced Service Discovery in Bluetooth," *Computer*, vol. 35, no. 6, pp. 96–99, 2002.

[18] C. Lee and S. Helal, "Protocols for service discovery in dynamic and mobile networks," *Int. Journal of Computer Research*, vol. 11, no. 1, pp. 1–12, 2002.

[19] A. ShaikhAli, O. F. Rana, R. Al-Ali, and D. W. Walker, "UDDIe: An extended registry for Web services," in *Proc. SAINT*, 2003, pp. 85–89.

[20] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *Proc. ISPA*, 2012, pp. 784–791.

[21] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *IEEE Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[22] A. ur Rehman Khan, M. Othman, S. A. Madani, and S. U. Khan, "A Survey of Mobile Cloud Computing Application Models," *IEEE Comm. Surveys and Tutorials*, vol. 16, no. 1, pp. 393–413, 2014.

[23] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[24] M. Golkarifard, J. Yang, Z. Huang, A. Movaghar, and P. Hui, "Dandelion: A unified code offloading system for wearable computing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 3, pp. 546–559, 2019.

[25] P. Liu, D. Willis, and S. Banerjee, "Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge," in *Proc. IEEE/ACM Symposium on Edge Computing (SEC)*, 2016, pp. 1–13.

[26] J. Garofalakis, Y. Panagis, E. Sakkopoulos, and A. Tsakalidis, "Web service discovery mechanisms: Looking for a needle in a haystack," in *Int. Workshop on Web Engineering*, vol. 38, 2004, pp. 1–14.

[27] A. Bröring, S. K. Datta, and C. Bonnet, "A Categorization of Discovery Technologies for the Internet of Things," in *Proc. Int. Conf. on the Internet of Things*, 2016, pp. 131—139.

[28] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, "Fogflow: Easy programming of iot services over cloud and edges for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, 2018.

[29] R. Klauck and M. Kirsche, "Bonjour Contiki: A Case Study of a DNS-Based Discovery Service for the Internet of Things," in *Proc. Int. Conf. on Ad-Hoc Networks and Wireless*, 2012, pp. 316–329.

[30] M. Antonini, S. Cirani, G. Ferrari, P. Medagliani, M. Picone, and L. Veltri, "Lightweight multicast forwarding for service discovery in low-power IoT networks," in *Proc. SoftCOM*, 2014, pp. 133–138.

[31] P. Gomes, E. Cavalcante, T. Rodrigues, T. Batista, F. C. Delicato, and P. F. Pires, "A Federated Discovery Service for the Internet of Things," in *Proc.Workshop on Middleware for Context-Aware Applications in the IoT*, 2015, pp. 25–30.

[32] S. Ben Fredj, M. Boussard, D. Kofman, and L. Noirie, "Efficient semantic-based IoT service discovery mechanism for dynamic environments," in *Proc. PIMRC*, 2014, pp. 2088–2092.

[33] V. Daza, R. D. Pietro, I. Klimek, and M. Signorini, "CONNECT: contextual name discovery for blockchain-based services in the IoT," in *Proc. ICC*, 2017, pp. 1–6.

[34] A. Zavodovski, N. Mohan, and J. Kangasharju, "edisco: Discovering edge nodes along the path," *CoRR*, vol. abs/1805.01725, pp. 1–6, 2018.

[35] J. Gedeon, C. Meurisch, D. Bhat, M. Stein, L. Wang, and M. Mühlhäuser, "Router-based brokering for surrogate discovery in edge computing," in *Proc. ICDCS Workshops*, 2017, pp. 145–150.

[36] D. Nguyen, Z. Shen, J. Jin, and A. Tagami, "ICN-Fog: An Information-Centric Fog-to-Fog Architecture for Data Communications," in *Proc. GLOBECOM*, 2017, pp. 1–6.

[37] S. Hoque, M. S. de Brito, A. Willner, O. Keil, and T. Magedanz, "Towards container orchestration in fog computing infrastructures," in *Pro. COMPSAC*, 2017, pp. 294–299.

[38] A. J. Fahs and G. Pierre, "Proximity-Aware Traffic Routing in Distributed Fog Computing Platforms," pp. 478–487, 2019.

[39] T. Li, R. Chandra, and P. Traina, "RFC1997: BGP Communities Attribute," Internet Requests for Comments, RFC, 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1997

[40] J. Heitz, J. Snijders, K. Patel, I. Bagdonas, and N. Hilliard, "RFC8092: BGP Large Communities Attribute," Internet Requests for Comments, RFC, 2017. [Online]. Available: https://www.rfc-editor.org/info/rfc8092

[41] R. M. Enger and J. K. Reynolds, "RFC1470: FYI on a network management tool catalog: Tools for monitoring and debugging TCP/IP internets and interconnected devices," Internet Requests for Comments, RFC, 1993. [Online]. Available: https://www.rfc-editor.org/info/rfc1470

[42] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding Traceroute Anomalies with Paris Traceroute," in *Proc. IMC*, 2006, pp. 153–158.

[43] C. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 97:1–97:37, 2019.

[44] F. Streibelt, F. Lichtblau, R. Beverly, C. Pelsser, G. Smaragdakis, R. Bush, and A. Feldmann, "BGP Communities: A Measurement Study," 2018. [Online]. Available: https://ripe77.ripe.net/presentations/40-communities_slides.pdf