Design, Resource Management and Evaluation of Fog Computing Systems: A Survey

Ismael Martinez, Abdelhakim Senhaji Hafid, and Abdallah Jarray

***Abstract*—A steady increase in Internet of Things (IoT) applications needing large-scale computation and long-term storage has lead to an over-reliance on Cloud computing. The resulting network congestion in Cloud, coupled with the distance of Cloud data centres from IoT, contribute to unreliable end-to-end response delay. Fog computing has been introduced as an alternative to cloud, providing low-latency service by bringing processing and storage resources to the network edge. In this survey, we sequentially present the phases required in the implementation and realization of practical fog computing systems: (1) design & dimensioning of a fog infrastructure, (2) fog resource provisioning for IoT application use and IoT resource allocation to fog, (3) installation of fog frameworks for fog resource management, and (4) evaluation of fog infrastructure through simulation & emulation. Our focus is determining the implementation aspects required to build a practical large scale fog computing infrastructure to support the general IoT landscape.**

***Index terms*—** Fog computing, fog design & dimensioning, fog resource management, fog infrastructure evaluation, simulation, Internet of Things (IoT), survey

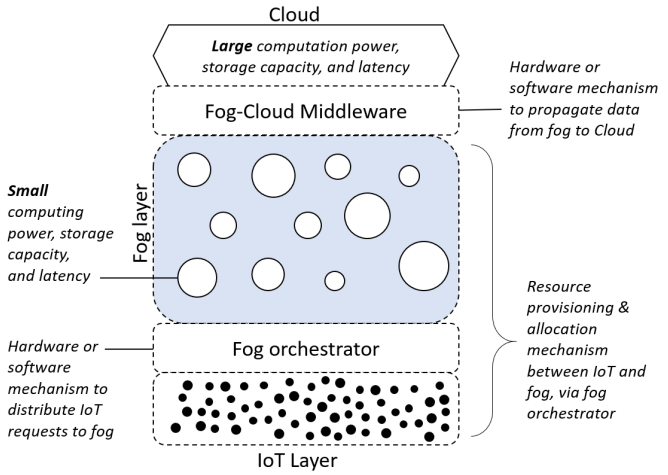


Figure 1: High-level necessary features and components for a complete fog system. Larger nodes have more resources.

I. Introduction

The benefits and varied use cases of Internet of Things (IoT) technology have led to an increase in IoT adoption, number of devices and applications, and volume of data uploaded to Cloud systems. International Data Corporation (IDC) predicts the number of connected IoT devices will exceed 41 billion by the year 2025, generating more than 79 zettabytes of data [1]. Current Cloud systems are not large enough to process and store this increase in IoT data traffic [2], an issue that affects all IoT systems. Congested networks towards a distant Cloud can result in relatively large delay for latency sensitive IoT applications such as health care [3], multimedia [4], and vehicular/drone applications [5, 6]. Furthermore, Cloud centralization can result in reduced privacy of uploaded IoT data [7].

Fog is a layer of geo-distributed servers with computing, memory, and network capabilities that serve as an intermediary between IoT and Cloud layers. Compared to Cloud, fog servers sit closer to IoT devices, providing reduced response time latency able to service most latency sensitive IoT applications [8]. Although fog servers are much smaller in terms of processing and storage capabilities than Cloud [9], the larger number and geo-distribution of fog servers allow fog to alleviate Cloud network congestion by servicing a large load of IoT applications [8]. Indeed, an IoT application can be fully serviced by local fog servers without propagation of IoT data to fog or Cloud further into the network.

Despite a substantial amount of research proposals in fog computing, there are very few documented implementations of fog in large-scale environments [10]. A design and implementation of a fog system requires several components of the fog layer, as well as collaboration mechanisms with IoT and Cloud. A detailed overview of these components and collaboration mechanism are presented in [11, 12, 13], and are summarized in Fig. 1.

We identify three broad stages in implementing and developing a fog system. First, in the absence of any fog system, a fog infrastructure is built through consideration of IoT service needs. Since large IoT traffic volume may cause increased network congestion towards Cloud, building a localized fog infrastructure close to high traffic areas is beneficial to internet service providers and Cloud service providers alike.

Second, the approach the fog infrastructure interacts with IoT and manages fog resources is defined. Fog resource management aims to select fog nodes to best process IoT data, and takes the form of algorithms or protocols implemented within individual fog nodes or a fog layer controller. In several cases, resource management relies on additional hardware/software structures (e.g. fog orchestration controllers

I. Martinez is with the Department of Computer Science and Operations Research, University of Montreal, Quebec, Canada H3C 3J7 (e-mail: ismael.martinez@umontreal.ca).

A. S. Hafid is with the Department of Computer Science and Operations Research, University of Montreal, Quebec, Canada H3C 3J7 (e-mail: ahafid@iro.umontreal.ca).

A. Jarray is with the School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, ON K1N 6N5, Canada (e-mail: ajarray@uottawa. ca).

or APIs) implemented within the fog infrastructure for appropriate fog node selection. Therefore, we consider the resource management algorithms/protocols, and the additional hardware/software structures separately. A *fog service provider* (FSP) is the organization that manages the implemented fog infrastructure, including IoT-fog interactions. Several proposed resource management approaches minimize service cost and latency to IoT users, which incentivizes IoT use of fog. Other approaches increase energy efficiency to reduce FSP operation costs. Therefore, efficient resource management for improved IoT service is an important concern to FSPs.

Third, evaluation of the constructed fog infrastructure with defined resource management protocols are evaluated to assess service impact on different IoT applications. Evaluation tools can be useful for IoT application developers to decide whether application computation is best done locally, by fog, or by Cloud. Tools for evaluation fog infrastructure use discrete-event simulation or emulation, and allow for configurable network conditions and IoT traffic patterns. To implement a fog computing system that can successfully support an IoT landscape, these three broad stages are focused into four key phases which will be explored in subsequent sections.

Phase 1: Estimate the volume of IoT traffic to be supported by fog, then **design & dimension** a fog infrastructure either from scratch, or by extending existing infrastructure.

Phase 2: Determine the method of fog **resource provisioning** for IoT use, and of IoT **resource allocation** to fog.

Phase 3: If necessary for resource provisioning, allocation and data migration, install a **fog framework** — additional hardware and/or software for fog resource management.

Phase 4: Use **fog evaluation** tools to measure the efficiency of the designed fog infrastructure and selected resource management approach in servicing IoT.

In this survey, we present a critical evaluation of solutions that contribute to the practical end-to-end implementation of fog computing. Our main contributions are as follows:

- Present and discuss existing models for fog design & dimensioning.
- Present a structured classification of resource management schemes based on initialization time and effectiveness for dynamic and static IoT applications. We also summarize optimization objectives and modeling techniques used.
- Review current framework solutions for fog resource management, and data migration, including an analysis of hardware and software overhead that is generated.
- Identify limitations of current simulation/emulation tools for the evaluation of fog infrastructures.
- Present open issues and research opportunities in practical fog implementation and evaluation.

Our discovery and selection of presented publications is meant to give a broad understanding of various proposed implementations of fog. We did select publications that cover the different facets of fog implementation. Indeed, publications are selected if they cover the following topics: (1) fog design; (2) fog resource management/orchestration; (3) fog evaluation, simulation or emulation; (4) fog applications; and (5) fog architecture. They are then categorized in the context of four phases. In this manner, we present a wide range of perspectives and approaches to fog implementation.

The remainder of this paper is organized as follows. Section II provides an overview of fog including motivation, architectures, and large-scale applications. Section III summarizes the limitations of current surveys in fog computing. Section IV reviews and compares current research in design & dimensioning of fog infrastructures. Section V categorizes resource provisioning & allocation schemes based on initialization time and effectiveness for dynamic or static IoT applications. Section VI surveys current proposals of fog frameworks for resource management and data migration across all fog nodes, and associated overhead. Section VII presents different simulation/emulation tools for the evaluation of fog infrastructures. In Section VIII, we discuss our findings, delineate the lessons learned and feature research challenges and opportunities for fog systems. Finally, Section IX concludes this paper.

II. Overview of fog Computing

We present an overview of the differences and benefits of fog to IoT when compared to Cloud and other edge technologies. Within the fog computing paradigm, we present several fog architectures that have been proposed to provide different ranges of IoT serviceability and inter-fog communication. Benefits and advantages unique to fog are particularly important to certain industries; hence, several industry applications of fog have been proposed for both local and large-scale implementations. Localization and geo-distribution of fog can provide increased privacy, but pose other security issues.

A. Definition of fog

Fog is a highly virtualized network comprised of nodes that provide processing and storage services to end devices (IoT) [8]. Fog nodes at the network edge, i.e. close to IoT devices, can provide computational support to IoT applications with minimal latency. Though typically at the network edge, fog nodes may appear hierarchically anywhere between the IoT layer and the distant Cloud [14]. This architectural setup allows many IoT requests to be satisfied by fog, and reduces the data volume reaching Cloud servers [15]. Since Cloud is still required by IoT applications that require high computation and long-term storage [16], the focus of fog is to support low resource and latency-sensitive IoT applications. According to [8], the main characteristics that define fog as a non-trivial extension of Cloud are: a) low latency and location awareness, b) wide-spread geographical distribution, c) support for mobility, d) very large number of nodes, e) predominant role of wireless access, f) heterogeneity, g) real-time interactions, h) interoperability and federation, and i) support for online analytic and interplay with Cloud. As a result, fog nodes can provide location awareness, activity awareness, time awareness, and energy awareness of IoT data processing [17].

Each fog node can be described as a small server or "micro data centre" [14] with available resources to support local computation. Although a typical fog node has reduced resource power compared to Cloud [9], the number of nodes in fog is

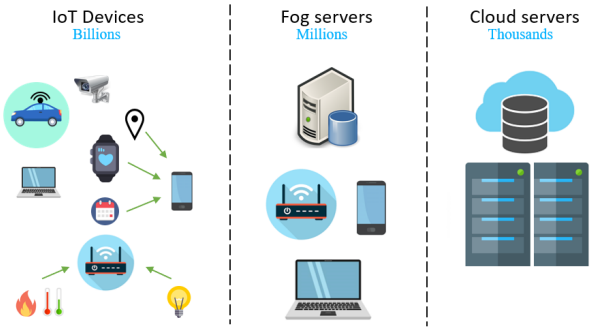
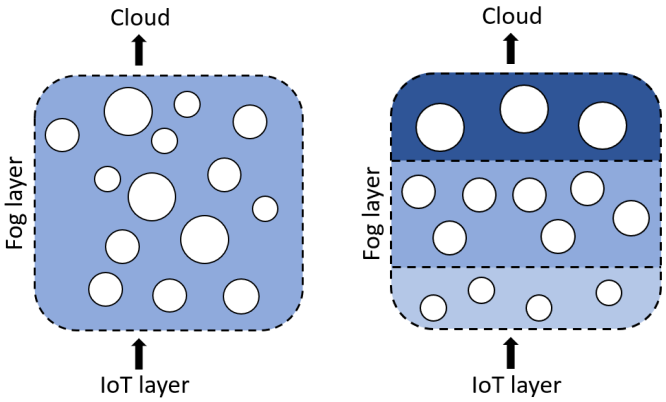


Figure 2: Examples and quantities of network devices per network service layer. IoT sensors may send data to a primary IoT device before data is transmitted and processed by fog/Cloud.



(a) Single fog layer hierarchy (b) Multi-tier fog layer hierarchy

Figure 3: IoT data is often transmitted to fog prior to being transferred to Cloud, therefore the IoT-fog-Cloud system is often viewed as a hierarchy. Within the fog layer, many routing protocols restrict data propagation to larger fog nodes (i.e. more resources), creating a multi-tier fog hierarchy.

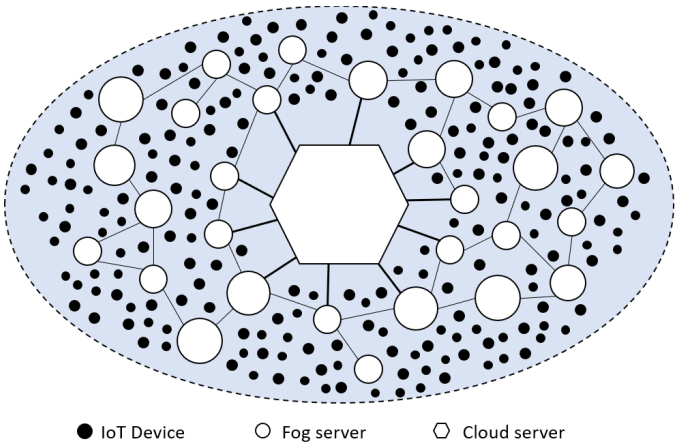


Figure 4: IoT-fog-Cloud physical network architecture. Larger nodes have more resources. Network links may be wired or wireless, and may involve intermediate routers/switches.

much larger [8]. The types and quantities of devices per system layer are described in Fig. 2. We summarize the difference between fog and Cloud in Table I. For the remainder of this paper, the terms "fog node" and "fog server" are used interchangeably.

Routing of data through fog often follows a path computing process [18] in which data is propagated to nodes of increasing size towards Cloud. Therefore, IoT, fog and Cloud layers can be expressed in a hierarchical architecture as in Fig. 3. Physically, fog servers are geo-distributed to be closer to IoT, providing the network structure in Fig. 4. Since each fog server has network connectivity, an IoT application may access any fog node either directly, or through a network access point.

A single physical IoT device may run multiple IoT *applications*. An IoT application that requires external computation or storage submits an IoT *request* to fog or Cloud. Each IoT request can be processed by one or more fog servers by partitioning into multiple *tasks*. Allowing for a request to be partitioned over several fog servers can result in parallelized execution of tasks, and decreased end-to-end response latency. We define a *static* IoT application as running on a fixed-location IoT device with frequent requests; otherwise, it is a *dynamic* IoT application.

B. Comparison of fog with other edge technology

Edge computing was introduced to bring storage and processing capabilities closer to IoT users for localized and low-latency computation [19, 20, 21]. Though the greatest benefits of edge computing occur when edge servers are in close proximity of data sources, edge computing is defined as being any computing and network resource between IoT users and cloud [19, 20]. Therefore, fog is seen as one implementation of edge computing [19, 20, 21]. Two other well-known implementations are *cloudlet computing* and *mobile edge computing* (MEC) [21].

A *cloudlet* is a small cluster of computers, forming a "data centre in a box" [22], and may be referred to as a *micro-cloud* [23] in some literature. Cloudlet computing refers to any implementation of cloudlets in the vicinity of end users. Cloudlets provide one-hop, high-bandwidth and low-latency wireless access to IoT users.

MEC, also known as *mobile cloud computing* [24], is a form of edge computing implemented within the Radio Access Network, and is therefore specific to mobile devices. MEC nodes are co-hosted at Radio Network Controllers or base stations, such as cell towers [25]. As a result, MEC nodes are always one-hop communication distance from active mobile users, and provide real-time processing of mobile requests. However, MEC does not provide edge services beyond direct network connectivity [25].

Fog nodes may be dedicated devices, but may also be legacy devices augmented with storage and computation resources [21]. This allows the fog infrastructure to be flexible since any device may become a fog node. However, fog nodes may be several hops from IoT users, and may require cooperation between multiple fog nodes for IoT processing. Although cloudlets and MEC nodes have no intercommunication, they are dedicated devices with high resource capabilities and direct

access to edge users. Hence, a cloudlet or MEC node is meant to process all IoT requests in full without further propagation. Cloudlets are depicted as being located within hotspot areas, such as hospitals or educational institutions [26], and MEC nodes are co-hosted with base stations [25]. Compared to fog nodes, MEC nodes and cloudlets may provide lower latency due to higher resource capabilities and one-hop proximity, but have lower implementation flexibility [21].

Across fog computing, cloudlet computing, and MEC, various service providers may own and manage a small subset of available devices. Without standardization of connectivity software, local IoT users may be limited to using a subset of nearby edge devices. When considering a large scale implementation towards Smart City technology, these restrictions can become a road-block for a fully connected digital ecosystem [27]. It is therefore encouraged for all edge computing to implement network protocols and software interfaces that provide unified IoT connectivity.

Our focus for the remainder of the paper will be largely on fog – i.e. systems where there exists intercommunication among nodes. We will use cloudlet and MEC when appropriate.

C. Fog Architecture

Many fog architectures in literature use a single layer stretching from IoT to Cloud layer, allowing any two fog devices to share data [3, 28, 29, 30]. This is the most general and flexible representation of the fog layer, with each fog node varying in distance from the edge and quantity of resources.

Intharawijitr et al. [31] propose a layer of horizontally placed fog nodes that cannot communicate with sibling nodes; instead, each fog node is restricted to communicate with only IoT and Cloud layers. In practice, processing latency can be reduced by having IoT upload data to the nearest fog node, and permitting fog nodes to migrate data amongst themselves if more resources are needed [28, 29].

Some fog architectures are represented as multi-tier hierarchies, with data sharing available across different fog layers but not within the same layer. Fog nodes are divided by computation power, memory, storage capacity, and proximity to IoT devices. IoT devices upload data to the first fog layer, which then uploads to higher layers until a fog node is found with sufficient resources. This architecture has been represented with two layers [32, 33] and three layers [34, 35]. iFogSim [36] defines a structure of multiple fog layers based on distance from Cloud, while [37] represents fog as a tree of fog nodes rooted by Cloud. Instead of tiers/layers of fog nodes, [29] and [38] define clusters of fog devices, with intra-fog and inter-fog communications. Similarly, [39] partitions fog nodes into clusters and has a hierarchy within partitions of the nodes.

Tang et al. [33] proposed a hierarchical architecture with Smart Cities in mind; the first layer sits at the network edge, the second layer is composed of larger fog nodes covering neighbourhoods, and the third layer uses largest fog nodes connected to Cloud to support city services. Arkian et al. [17] develop crowd-sensing applications supported by IoT and fog for small scale city services such as parks. Sun and Ansari [40] propose a hierarchical extension to MEC that connects to fog. Each cellular base station is connected to the fog infrastructure to alleviate edge traffic and handle large mobile data streams.

D. Motivation

The large incentive of using fog computing is its ability to process IoT data with real-time or latency-sensitive requirements. Use cases that benefit from fog cover health care, autonomous vehicles, and multimedia.

1) Health Care: Gill et al. [41] propose to use body sensors with fog to help diagnose heart disease. It has also been proposed to use fog with wearables and sensors [3, 28] to provide real-time assisted living services to patients in hospitals or health care centres.

2) Autonomous Vehicles: Loke et al. [6] proposed an asset management concept for autonomous drone technology. A fog server can provide control signals for drone navigation, given a line-of-sight between a drone and the fog server. Fog servers can also relay traffic condition information from smart vehicles [5, 29], creating area-wide, real-time traffic sharing; this ultimately reduces road accidents. Coupled with smart traffic lights [42], fog increases the efficiency of route navigation.

3) Multimedia: Due to the per-instance and real-time processing fog can provide, fog has been proposed to deliver processing to multimedia, such as gaming, video streaming, and augmented reality [4]. Video surveillance applications can use fog for facial recognition, diminishing the response time of appropriate authorities in the event of an incident [37, 43]. Surveillance cameras at the scene of an incident can create bursty data; a decentralized fog infrastructure can process different data in different fog nodes, resulting in an overall quicker emergency response [43].

For IoT applications in health care, smart vehicles and multimedia, a difference of milliseconds in response time can lead to a significant impact on event outcome. In these cases, fog computing can provide overwhelming benefits in supporting IoT applications at the network edge.

E. Industry Applications

Applications of fog computing systems have been proposed in several industries to take advantage of the unique traits of fog. Applications range from local to large-scale implementations.

1) Internet of Vehicles: Fog and road-side units (RSUs) can collect traffic information from smart vehicles. Integrating fog with an Internet of Vehicles (IoV) infrastructure can allow cars to participate in sharing real-time traffic conditions throughout the city [5, 29]. Other fog applications of IoV infrastructure use clusters of slow moving or parked cars as the fog itself [44, 45]. Sookhak et al. [45] propose incentives for participating such as free parking, free Wi-Fi or free shopping vouchers, while Hou et al. [44] show how non-smart cars may be upgraded with hardware and/or software in order to take part. For fairness and participation motivation, it is proposed that incentives for participating vehicles correspond to the quantity

and type of contributed resources [46, 47], and provide privacy & security to vehicles and users [46].

2) Health Care: Santos et al. [3] argue the use of fog for e-health monitoring systems by providing a stochastic analysis of fog server reliability when backed by Cloud. They conclude that fog-Cloud system failures are small enough (under 0.3%) that they do not nullify the benefits of fog. Ahmad et al. [48] consider the use of fog to provide better control of data privacy & security of smart phone health applications.

Gateways act as intermediaries between sensor networks and Cloud systems; Rahmani et al. [49] envision the use of fog-enabled *smart e-Health Gateways* to support local computation and storage for body-worn or implanted sensors in a smart hospital or home. Medical cyber-physical systems (MCPS) provide seamless connection between healthcare devices and computational resources. Gu et al. [28] introduce a fog infrastructure to support MCPS.

3) User Provided Fog: More generally, *Consumer as a Provider* platforms allow for user devices such as phones and modems to act as fog devices and are made available to the public [50, 51]. With a large enough user base, this fog infrastructure can become very large in scale covering a wide area such as a city.

4) Smart Cities: The concept of a Smart City is to improve the life of citizens through integrated monitoring and adaption of city services [52]. Examples include Smart Grid, Smart Transportation, and crowd-sensing applications.

Smart Grids are electricity networks updated with smart meters and shared customer usage information to service providers. This incoming information determines how much electricity should be generated and where it should be sent. Okay and Ozdemir [53] propose the use of fog computing for scalable real-time electrical usage monitoring and improved privacy of information sharing.

Smart Transportation applications attach IoT sensors to public transportation such as buses and subway trains to share real-time information on transport location and delays [54]. Road-side and fog computing infrastructures can facilitate optimal route calculations and collision avoidance for smart cars, including self-driving cars, via real time traffic conditions over connected vehicles [5, 29]. Smart traffic lights optimize traffic by flow using vehicular sensors and traffic cameras [55].

Bittencourt et al. [56] propose a fog architecture that provides real-time IoT application allocation and processing, especially suited for mobile IoT such as smart phones and smart cars within a city. Non-invasive and low-cost static sensors can be set up in densely populated public areas to provide real-time crowd-sensing services when used alongside fog [17]. Installed in an outdoor park setting, crowd-sensing devices can passively determine in which areas the most activity is taking place, resulting in awareness of possible maintenance or updated amenities. This concept can be extended to other Smart City use cases such as monitoring air pollution or noise pollution from mounted sensors on outdoor and indoor public transportation respectively.

Unmanned Aerial Vehicles (UAVs) can be quite useful in providing additional computation services to IoT in a Smart City environment. One approach is to dispatch UAVs over environments of large IoT traffic to provide direct IoT support [57, 58]. Since these UAVs exhibit cloudlet and MEC functionality, they are designed to process the full amount of IoT resource requirements without propagation, which may create large resource and energy demands on UAVs. A more robust fog-enabled approach is to dispatch a UAV over low service areas and connect to the surrounding IoT and fog infrastructure [59]. Low service areas may be a result of abnormally high IoT traffic, a failed fog node, or areas that are challenging for human or manned vehicle access.

More city focused use cases include Smart Agriculture, Smart Health & Well-being, Smart Waste Management, Smart Water Management, Smart Greenhouse Gas Control, Smart Retail Automation [60], Smart Pipeline Monitoring [33], Noise Pollution Mapping, Urban Drainage Networks [10], augmented reality [15], City Structure Health Monitoring, Environmental Monitoring, and Public Safety & Security [55]. Together, they become components of a sustainable Smart City supported by fog for real-time information queries [55, 60].

F. Privacy & Security

Fog provides computation and storage resources over servers that are geographically distributed, providing a means to isolate IoT data computation and/or storage to localized fog servers. This layout allows sensitive IoT data, such as health care, to never leave the vicinity, keeping the data from being collected and used by unwanted parties such as tech giants [7].

The distribution of fog nodes and servicing of heterogeneous IoT can lead to security issues between nodes. Dzousa et al. [61] propose a policy-based management framework to support secure communication, collaboration, and interoperability of requested resources in fog. Liu et al. [62] use hash puzzles distributed to nearby vehicles to eliminate possible denial-of-service attacks to smart traffic light systems with fog capabilities.

Network security and congestion can pose a problem to fog in providing low-latency services. The CloudWatcher framework [63] uses OpenFlow [64] to monitor the network for intrusion detection and other security risks.

The majority of research into privacy-preserving communication and data security uses homomorphic encryption from IoT, or attribute-based encryption between an IoT-fog pair [65]. Since the focus of this survey is in the implementation of fog regardless of IoT behaviour, we do not further discuss privacy & security issues in fog beyond an awareness of their existence.

III. Existing Surveys on fog Computing

Hu et al. [66] explore the characteristics and benefits of fog when used with IoT and Cloud. They present a comparison between Cloud computing and fog computing paradigms. They also present an in-depth description of computation, storage, and communication technologies used in fog. Dolui et al. [21] discuss the concepts, benefits and technologies of edge computing. They provide a detailed comparison of the three main paradigms of edge computing: fog computing, cloudlet computing, and mobile edge computing (MEC).

Table I: Differences between fog and Cloud layers.

Feature	Fog	Cloud
Latency	Low	High
Distribution	Geographically distributed	Centralized
Distance to network edge	Close	Far
Number of nodes	Millions	Thousands
Resource size	Small	Large
Access	Predominantly wireless	Wired and wireless
Heterogeneity	High	Low
Interaction with IoT	Real-time	Batch processing
Owned & Managed	Various service providers	Few large organizations

Mukherjee et al. [67] study advancements and benefits derived from integrating fog into current technologies, such as virtualized fog data centres, fog radio access networks, and software-defined network (SDN) enabled fog architectures. Resource allocation models and techniques are discussed alongside mathematical models of fog components such as latency, energy consumption, and resource sharing.

Mouradian et al. [68] present a comprehensive review of major contributions in fog covering six criteria of heterogeneity, QoS management, scalability, mobility, federation, and interoperability. Ghobaei-Arani et al. [69] provide a systematic and comprehensive literature review of resource management issues and solutions in fog computing. They classify mechanisms and techniques into application placement, resource scheduling, task offloading, load balancing, resource allocation, and resource provisioning. Brogi et al. [70] present an exhaustive overview of resource allocation solutions within fog. Surveyed contributions are further classified based on an algorithmic perspective which looks at solution methodology, and a modeling perspective which looks at constraints and optimization metrics. Neither [68, 69, 70] consider the time overhead of resource provisioning & allocation prior to IoT processing. Indeed, though the assigned fog servers to an IoT application may provide optimal latency, the assignment process may be too slow for time-sensitive IoT applications.

Naha et al. [71] review the publication trends of fog computing and Cloud computing alike, and present a taxonomy of fog research publications by requirements of infrastructure, platform and application. They provide an overview of other technological architectures analogous to fog such as edge computing and dew computing. Mahmud et al. [11] identify key challenges and properties of fog computing, and use them to provide a taxonomy of aspects in fog computing such as fog node configuration, nodal collaboration, service level objectives, applicable networking system and security concerns. Ahmed et al. [12] select and review 30 actual or proposed fog applications. The selected contributions have little overlap, and cover a broad range of industries, uses and communication methodologies. Selected reference applications are used to study reasons for using fog, required fog hardware platforms, assumed data distribution methods among fog, leveraged fog service models, privacy & security requirements, and application workload characteristics on fog. Yi et al. [72] describe the issues potentially faced when designing and implementing a fog system, such as in IoT communication interface, computation offloading, accounting, and resource management. Yi et al. [65] describe the privacy & security issues that arise from IoT-fog communication. Geo-distributed and edge location features of fog can expose an IoT device's location to a small radius around the connected fog node, in addition to possible exposure of application data and usage frequency to fog. From the IoT perspective, the owner of a fog node is not always evident, resulting in trust and security issues when connecting to an arbitrary and close fog server.

Markus and Kertesz [73] provide a taxonomy of simulation tools and environments for fog and edge computing. They propose a taxonomy of available simulators modelling fog, edge, Cloud and IoT networks to aid researchers in distinguishing the right tool for different research needs.

Across these surveys, open issues and research challenges in fog computing are discussed [11, 12, 66, 67, 71, 72], applications of fog computing to IoT use cases are summarized [12, 66, 67, 68, 71], and gaps in current research towards future work are identified [66, 71, 72]. Some survey review individual components of the fog computing system such as resource provisioning & allocation and fog frameworks [68, 69, 70], security & privacy [11], and fog simulation software [73]. All discussed open issues address algorithmic enhancements to resource management techniques of existing fog infrastructures. Additional hardware may also mitigate these issues at the cost of generated overhead. However, changes and additions to fog design are not covered. Our survey, on the other hand, provides a holistic view of the applicability, challenges, overhead, and limitations of proposed fog systems, irrespective of fog technology used.

To conclude, we summarize the limitations of existing surveys as follows: (1) none covers the four phases to realize fog systems; (2) none covers design & dimensioning of fog systems; (3) none analyzes the generated overhead of framework implementations; (4) none covers the multiple migration scenarios between fog servers; (5) none considers resource provisioning allocation time overhead prior to IoT request processing [68, 69, 70]; and (6) none considers changes to fog design to mitigate open issues.

In this paper, we intend to detail the full fog implementation process, beginning with an IoT environment without any available fog system. We review and compare the current contributions for designing & dimensioning a fog infrastructure. We identify efficiency, assumptions, shortcoming, and generated overhead of resource provisioning & allocation schemes, and fog frameworks. We review the features and efficiency of simulation/emulation tools for the evaluation of a designed fog infrastructure, implemented resource provisioning & allocation mechanism, and conceptualized fog framework. Finally, we identify the limitations and open issues of all components of fog implementation.

IV. Fog Design & Dimensioning

For users and organizations wishing to implement their own fog infrastructure to support local edge devices, Mahmud et al. [11] outline the ground work for what components and mechanisms are necessary in fog (see Fig. 1). This does not however give insight into the location and quantity of installed resources of each fog node, known as *design* and *dimensioning* respectively.

Design of edge networks to provide low-latency access and processing to IoT has been studied with cloudlet computing, which involves no intercommunication between cloudlets [74, 75, 76, 77]. In most cases, cloudlets are designed to be one-hop away from IoT devices; however Ceselli et al. [76] propose an augmentation to MEC where data is routed from base stations to a nearby cloudlet. Though other contributions are only interested in network placement of cloudlets, Fan and Ansari [77] dimension the number of cloudlet servers installed in each designed cloudlet location. Building a dedicated computing infrastructure with high resources may be costly, and underused in most cases. A fog system provides more network flexibility and geo-distribution of smaller devices, potentially covering and servicing a much wider IoT ecosystem. To our knowledge, there are currently only two contributions that develop an optimal fog design scheme. Both schemes optimize fog node locations and installed computing & memory resources while satisfying IoT QoS requirements.

Yu et al. [5] consider fog to provide real time processing for autonomous vehicles. They propose a fog design & dimensioning scheme of RSUs, fog nodes, and Internet Gateways which work together to provide real-time traffic information to enhance and facilitate automated navigation and collision-avoidance. Given a set of candidate locations, candidate resource configurations, and a known number of connecting vehicles for a certain location and time period, the location and resource amounts of RSUs, fog nodes and Internet Gateways are optimally found via a Mixed Integer Linear Program (MILP) in an arrangement that minimizes infrastructure costs. RSUs may be fog nodes themselves (coupled variant) or are separate from fog nodes (decoupled variant); both variants are tested and compared. They conclude that a decoupled model allows design flexibilities that result in a more economical and cost-effective scheme. For scalability, a heuristic algorithm based on the decoupled model is used.

Regarding vehicular traffic, the model assumes a known static set of vehicle resources accessing the network across different regions, which may not be true in practice. Although having a set of candidate locations for fog nodes is practical, the model also assumes a finite candidate set of dimensioning configurations. The solution to this model is thus dependent on the completeness of such a set. Finding the optimal placement of RSUs and Gateways increase the complexity of the model, while the inclusion of RSUs also restricts the application of this model to IoV.

Martinez et al. [78] propose a fog design & dimensioning scheme to support the general IoT landscape. For a given area, the future IoT data volume and the resulting stochastic network congestion distributions are estimated, which affect the approximated IoT-fog end-to-end communication delay. An IoT request with k tasks is represented by a Task Dependency Graph of k nodes, and the set of physical candidate fog node locations are represented by a bidirectional graph. Tasks and task dependency links are mapped to physical fog locations and fog infrastructure paths respectively. A fog design & dimensioning scheme is defined to find a mapping that satisfies fog node resource capacities, fog infrastructure bandwidth capacities, and IoT QoS requirements.

An MILP model (fog-DC-MILP) is used to find an exact optimal fog infrastructure by minimizing infrastructure deployment costs. Due to the intractability of the fog-DC-MILP model, a Column Generation model (fog-DC-CG) is proposed.

Simulation and scalability testing between fog-DC-MILP and fog-DC-CG show a significant reduction in solution computation time of fog-DC-CG with near-optimal cost. Furthermore, fog design & dimensioning solutions of both models are similar. This indicates fog-DC-CG is a practical alternative model to fog-DC-MILP.

For each candidate fog node, they define a maximum amount of each resource that can be installed, allowing for resource configurations selected from a continuous set. This aims to remove the concern of discrete resource configuration set completeness observed [5]. The designed & dimensioned infrastructure [78] is extensible, allowing for current fog nodes to be upgraded or extra fog nodes to be added to the current infrastructure if IoT data volume increases.

They use a discrete set of IoT devices, each uploading data at a rate following a Poisson Process. They compute a percentile estimation of expected IoT traffic and resulting congestion to produce a deterministic upper bound. This allows for simple modifications to beginning percentile parameters to increase or decrease traffic estimation, resulting in a change in the fog design & dimensioning solution.

Due to the models extensibility, it is reasonable to make an underestimate of expected IoT traffic and extend the designed & dimensioned infrastructure based on future performance. These two contributions [5, 78] are summarized and compared in Table II.

V. Fog Resource Provisioning & IoT Resource Allocation

Fog *resource provisioning* refers to the reservation of computational and memory resources within fog nodes for use by IoT applications. IoT *resource allocation* refers to the assignment of resource requirements for an IoT request to the fog. It is clear that resource provisioning and resource allocation are two sides of the same coin, since a fog node needs to provision its resources for the allocation of IoT requests. For all resource provisioning & allocation schemes, several consistent infrastructural assumptions include: 1) existence of a pre-defined fog infrastructure, 2) all fog nodes are reachable and available from any IoT device, 3) any pair of IoT device and fog node experiences static network congestion and/or latency, and 4) uploaded data format and response is homogeneous.

Table II: Description of contributions in fog design & dimensioning.

Scope	Yu et al. [5]	Martinez et al. [78]
Main Contribution	Scalable design & dimensioning for fog nodes, RSUs and Gateways to support autonomous vehicles.	Scalable fog design & dimensioning to support general IoT systems with near-optimal implementation cost.
Supported IoT devices	Smart Vehicles.	General IoT devices.
Pre-defined device candidates	Fog nodes, RSUs, Gateways.	Fog nodes.
Predicted IoT traffic	Discrete vehicular traffic set.	Percentile of stochastic IoT traffic predictions.
Resource dimensions	Selected from discrete pre-defined set.	Continuous up to a maximum capacity.
Extensible	—	Fog nodes can be added to current fog infrastructure to account for increased IoT traffic.
Congestion	—	Accounts for possible network congestion by designing fog under worst-case network scenarios.

For IoT applications that intend extended use by the same fog servers, the server partitions a *module* of reserved resources. Once a module is set for an IoT application, all requests from that application are immediately processed by the module for reduced long-term latency. However, the reservation of fog resources itself may take time, and may not be useful for IoT applications which require immediate and infrequent processing. Module *migration* is the process of freeing module resources in the current fog node, and re-provisioning the module in a different fog node or Cloud. Any IoT processing data or storage present in the current fog node is transferred to the new module.

The allocation schemes [9, 31] assume each IoT is comprised of a single task, while Agarwal et al. [30] propose to process each request by a single fog node, split into multiple tasks if there are insufficient resources. In both cases, the entirety of the IoT data need only be processed by a single fog node. Instead of mapping each IoT request separately, Yousefpour et al. [43] go further by clustering IoT devices together that run the same service, and map those services to fog node modules. Since all computation of an IoT application is done on a single fog server, many requests can result in a large processing queue and high latency.

Taneja et al. [79] report that each IoT request is in fact comprised of multiple tasks — stemming from multiple sensors and actuators — that are too taxing to be processed on a single fog node. Therefore, an IoT application's multiple tasks are split among one or more fog nodes. The multiple tasks of a single IoT request are often depicted as a directed acyclic graph, with each directed link representing a dependency between tasks [79, 80]. Hence, fog processing of tasks may require the same task workflow order. Similar approaches are to allocate application tasks to different fog nodes either via algorithms [37, 79, 80], policies [16, 36, 56] or optimization models [32].

In our review of current literature in this space, we did observe that all proposed models fell into one of three classifications based on prompt or optimal service to IoT applications. The fog layer may reserve resources for future IoT use based on accurate IoT traffic predictions. When IoT requests arrive, it is assumed the resources are available and immediate processing takes place. Schemes that follow this approach are known as *prior provisioning and prompt allocation* schemes.

Since IoT traffic predictions require additional computation effort and may be faulty, most schemes will provision resources only once IoT requests have arrived. There are two general approaches to resource allocation using on-demand provisioning: prompt allocation and small batch allocation. When an IoT allocation is promptly serviced, it is sent to the nearest fog node for processing, regardless of cost. Models that follow this approach are known as *on-demand provisioning and prompt allocation* schemes, and are ideal for dynamic IoT support. Another approach is for a fog resource manager to accumulate a small batch of IoT requests, and find the optimal allocation of IoT tasks that optimize some metric (e.g. latency, resource cost). Once a module is provisioned for an IoT request, all future requests from the same IoT application are provided immediate service by the module. Hence, small batch allocation provides efficient IoT processing for the life-span of the provisioned module. Models that follow this approach are known as *on-demand provisioning and small batch allocation* schemes. Though these schemes provide more efficient long-term IoT support, execution of these schemes are significantly slower than prompt allocation methods. Therefore, small batch allocation schemes are better suited for static IoT applications. A summary of these three classifications of resource provisioning & allocation schemes is provided in Table III and Fig. 5.

A. Prior provisioning and prompt allocation

Aazam and Huh [14, 81] propose the analysis of fog resource usage data of IoT devices to determine the relinquish probability a new service request will be abandoned within a time frame. IoT devices with low relinquish probability, i.e. will continue fog usage for long periods, are offered slightly lower usage prices as well as higher allocated resources by fog. Behavioural analysis of previously connected IoT ensures that enough resources are reserved by fog for future predicted IoT traffic, and allows for an immediate connection and processing of IoT data upon a new request. For IoT devices that have never connected prior, a default low relinquish probability is assumed, and resource pricing and quantity are calculated accordingly in real-time. This concept is extended in [82, 83]

Table III: Process of resource provisioning & allocation schemes by prompt or optimal service.

Resource Provisioning	Resource Allocation: Prompt	Resource Allocation: Small Batch
Prior	• Reserves resources in fog based on historic predictions of future IoT traffic. • IoT requests are allocated immediately to reserved resources. [14], [81], [82], [83]	NA
On-demand	• An IoT request arrives to a fog node; the fog node verifies if it has sufficient resources. • If yes, it processes the IoT request. Otherwise, it propagates request to a further node. [29], [43], [30], [47], [56], [84], [85]	• Groups several IoT requests for batch resource provisioning & allocation. • IoT requests are distributed in fog to optimize efficiency. [4], [9], [17], [28], [31], [32], [79], [80], [86], [87], [88], [89], [90], [91], [92]

to also consider historical quality of experience (QoE) of IoT based on end-to-end delay, jitter, packet loss, latency, and blocking probability. This approach more efficiently predicts future resource consumption for real-time allocation for multimedia [82] or haptic sensors [83].

Although an IoT application's QoS is satisfied, QoE may be low and thus requires different fog node connections or more fog resources on future requests. Instead of requiring the allocated fog node to satisfy the required latency of the IoT application, [14, 81, 82] focus on providing high fog utilization assuming any fog node could satisfy the IoT latency requirements. On the other hand, [83] ranks potential fog nodes for IoT allocation by latency, and verifies the latency suitability of the fog node before assigning resources.

B. On-demand provisioning and prompt allocation

Agarwal et al. [30] propose a resource provisioning scheme that does not rely on any current or historical information from IoT or fog. The proposed algorithm begins with an IoT application sending its request to an arbitrary fog node within communication range, usually the nearest node. If the fog node has enough resources to process the request, it will do so; otherwise, the request is partitioned into several tasks, and are sequentially processed by the limited fog resources. If no resources are available in that particular fog node, the IoT request is propagated to Cloud. In this worst case scenario, the propagation to Cloud may result in high latency and unsatisfied IoT QoS. Intermittent fog resource sharing among the fog could allow the initial fog node to know which other fog nodes have enough resources [93], and forward accordingly thus keeping latency low.

Bittencourt et al. [56] extend this idea by introducing three possible fog processing policies. In all cases, an IoT application is assigned to the first fog node with which it connects, but the processing order of IoT applications differs. If no resources are available at the fog node, Cloud propagation is applied. The *concurrent* strategy performs fog processing on IoT data regardless of current available resources; applications within a fog node are processed in parallel, and the allocation indifference to current resources could lead to high processing latency. The *First-Come-First-Serve* (FCFS) strategy processes IoT requests in the order of their arrival to a fog node. The *delay-priority* strategy processes the IoT application with the lowest QoS latency requirement first, and re-orders the next IoT application to process as new requests arrive. Simulations show that the FCFS and delay-priority strategies yield the lowest latency, whereas the concurrent strategy yields the lowest amount of data transferred to Cloud.

For vehicular fog settings where mobile vehicles require computation from static or slow-moving vehicles, Peng et al. [84] propose a multi-attribute double auction mechanism for base stations to match and pair vehicular fog nodes with vehicular IoT users. The mechanism allows vehicular fog nodes to announce their resource attributes, reputation and asking price, which is met with IoT announcements of resource and latency requirement, and bidding price. The formulated one-to-one assignment algorithm for resource matching executes in under 8 milliseconds for up to 100 vehicular fog nodes and up to 100 vehicular IoT devices. This scalable matching algorithm adds very little to the overall near instant computation provided by the allocated vehicular fog node. Similarly, Zhou et al. [47] introduce a contract-based mechanism for IoT request offloading to nearby smart vehicles. A contract is designed and offered to a vehicle based on the amount of resources and time in return for a reward; IoT users requiring fog computation are paired to a vehicular fog via a pricing-based stable matching algorithm.

Zhang et al. [29] introduce a cooperative fog computing architecture to deal with big IoV data. It allows for data migration between fog nodes for mobile smart cars. As a result, two separate resource allocation strategies are considered based on available resources at the nearest fog server. Each fog node in this system has a finite set of Virtual Machines (VMs) which partition fog resources for IoT use. If the number of VMs in a fog server is sufficient to process an incoming IoT application, intra-fog resource management will allocate the application to VMs that minimize fog energy-consumption via convex optimization. If data migration is required, a min-max optimization model is used to determine to which fog node data should be transferred to minimize the transfer rejection probability. The delay of data transfer between fog nodes is measured at 30 – 70 ms, providing low additional delay to the

allocation process.

Yousefpour et al. [43] address the problem of dynamically deploying or releasing IoT services on fog nodes by means of two possible greedy algorithms that comply with QoS requirements. Both algorithms determine periodically from which fog node modules should be released and transferred to Cloud, in order to free resources for future IoT requests. The min-cost algorithm aims to allocate IoT services to the fog node that would provide the lowest resource allocation cost; it similarly releases IoT services that incur large resource costs. The min-viol algorithm allocates IoT services with high demand and releases services with low demand from fog nodes. The response latency inherently increase for modules released to Cloud, and thus results in a QoS violation.

Xia et al. [85] propose two ordering-based heuristics to search and select fog nodes to which an IoT application is allocated. With *anchor-based fog node ordering* (AFNO), fog nodes are ordered by its latency to an anchor IoT application. *Dynamic component ordering* (DCO) attempts to allocate IoT tasks until a constraint failure in the search occurs; the components are reordered with the failed tasks being allocated first, and may require several reorders for a successful application allocation. Ordering fog nodes by latency adds sorting overhead to AFNO, resulting in lower execution times with DCO allocation. For up to 20,000 fog nodes, DCO executes in under 100 ms for a single IoT application, allowing for real-time resource allocation and execution.

Reinforcement learning (RL) is a machine learning technique that aims to make improved decisions over time based on the reward or penalty incurred by previous actions [86]. In many RL approaches to resource allocation [87, 88, 94], this training process is divided into two parts: 1) receive IoT request, feed through RL model and take action, and 2) update RL model based on action reward/penalty. This process prioritizes the resource allocation decision, providing real-time support to IoT. Since RL techniques improve with time, the RL models are initialized with random values to start, meaning the initial resource allocation decisions may not be optimal. If the RL model is trained with batches of IoT requests, then the request data is saved in memory until enough requests are accumulated for training [94]. Since RL can be computationally taxing, it is often proposed to use a separate fog server to conduct all RL modeling [87, 88].

Sun et al. [94] use power consumption of fog nodes as a reward/penalty to model an energy-efficient resource allocation scheme. The action taken defines which fog processors to turn on or off, and to which layer to send an IoT request (fog or Cloud). Wei et al. [87] use a single fog server to compute all RL model updates, which receives and distributes IoT requests to the remaining fog servers. The formulated model aims to optimize response latency of resource allocation and content cashing.

Wang et al. [88] propose a RL model for fog allocation in IoV environments for minimizing response latency. All IoT requests are sent to an independent and centralized fog sever, where all RL model updates are computed. The fog server returns a decision to the mobile device of which layer to access: cellular network, device-to-device network, or fog network. Then, the mobile device sends another request to the appropritate network for processing. This procedure requires communication with two separate nodes which may increase latency. Furthermore, although QoS requirements of IoT, energy consumption of fog and total latency are considered, the resource capacities of each fog node are not. It is assumed that each layer can always process incoming IoT requests, and the main consideration is the best distribution to do so.

C. On-demand provisioning and small batch allocation

Zeng et al. [89] research fog supported Software Defined Embedded Systems with client-side computation support. In this scenario, client-side computation incurs a certain cost with client-side computation latency, whereas fog incurs a different resource cost, and both transmission and computation latency. For each IoT device, computation placement (client-side or fog) is formulated as a Mixed Integer Non-linear Program (MINLP) to minimize overall processing latency. This MINLP is linearized, and re-formulated as a three-stage heuristic. Although it may be convenient to assume an IoT device has client-side processing capabilities, this assumption is not practical for a general IoT ecosystem.

Souza et al. propose a resource provisioning scheme where fog node resources are partitioned into *slots* of fixed size, with IoT requests requiring a set number of slots. In particular, it is assumed that an IoT request either requires a small number of slots and can thus be processed anywhere in the fog, or requires a large number of slots and can thus only be processed by second-tier fog nodes. Formulated as an MILP, this resource provisioning simulation setup is wildly simplistic and requires more dynamic parameters to approximate a practical fog system scenario.

Zhang et al. [90] introduce *massive data centre operators* (MDCOs) as a middle-man between fog and IoT. A Stackelberg game is played between fog and MDCOs, and MDCOs and IoT devices to determine an optimal resource price and amount provided by fog and MDCO to maximize fog utilization.

Ali et al. [92] propose a many-to-one matching game between a set of fog nodes and IoT devices. Each device will rank a potential pair based on perceived latency and utilization, and are subsequently matched with their highest feasible choice. Although the matching algorithm itself is quick, node discovery followed by utilization and latency calculations for ranking cannot be done in real-time. Therefore, this method is ideal for small batch allocation.

When an IoT request is assigned to a fog node with insufficient available resources and is immediately propagated to Cloud, it is said that the request is *rejected* by the fog node. Assuming that fog may not have enough resources to service all IoT devices in the region, Intharawijitr et al. [31] formulate a model that minimizes the *blocking probability* of resource allocation, which is the average number of IoT workloads rejected by fog. For an IoT workload, three fog selection policies are explored: a *random* fog node, the fog node with *lowest-latency* to the application, or the fog node with *maximum available capacity*. Simulation results conclude

the lowest-latency policy minimizes the blocking probability of IoT applications.

Skarlat et al. [9] formulate resource provisioning over the fog-Cloud system as an MILP to maximize fog utilization. The simulated evaluation concludes a decrease in response latency by 39% over default provisioning policies in CloudSim [95]. The same problem is formulated as a genetic algorithm with an evaluated increase of 150% cost of the MILP solution, and around 64% of applications assigned to fog.

Taneja and Davy [79] model each IoT application request as a set of dependent IoT tasks that may be processed by different fog nodes. The resource requirements of all current batch IoT tasks are sorted in ascending order, and a fog node satisfying the task resource requirements is selected for each task. By efficiently placing IoT applications on fog, increased fog utilization implicitly decreases the total response time, network usage and energy consumption compared to Cloud placement strategies. Karamoozian et al. [80] solve the resource allocation problem using a gravitational search algorithm. This meta-heuristic algorithm denotes each possible solution with a mass, and iteratively updates the particle masses based on their interactions with each other. At termination, the heaviest mass (solution) is selected.

Whereas [79, 80] represents each IoT application as a directed acyclic graph of IoT application tasks, Ni et al. [96] represent IoT tasks as a priced-timed Petri net which includes task transition time and price costs, in addition to task dependencies. The presented heuristic algorithm [96] allocates IoT applications to fog by prioritizing the minimization of response latency, followed by the maximization of fog *credibility* based on time cost, price cost, and resource capacity. The fog credibility is based on historical data of response rate, execution efficiency, reboot rate and reliability.

The focus of resource provisioning by Salaht et al. [97] is to first and foremost satisfy QoS requirements of IoT applications without further consideration of fog utility, resource cost or latency. Formulated as a Constraint Programming problem [97], it aims to create a scalable, generic and easily upgradeable placement service. Using the Smart Bell application [85] for comparison, the MILP model [85] executes in 343 seconds, whereas the Constrained Programming model [97] executes on the same application with the same parameters in 0.559 seconds without loss of solution QoS. We observe that considerations of cost optimality are ignored for a faster solution which may still prove too slow for real-time allocation of latency-sensitive IoT applications.

Donassolo et al. [98] introduce *Optimized Fog Service Provisioning* (O-FSP) resource allocation scheme to minimize IoT service cost and increase fog utilization. Based on available fog node resources and IoT QoS requirements, O-FSP follows a greedy approach to allocate IoT applications by resource cost. Although O-FSP succeeds in providing high fog infrastructure utilization and low resource costs, the execution time of O-FSP is not evaluated.

Gu et al. [28] support medical systems by reserving fog resources in the form of a VM assigned to the medical device. The resource provisioning problem is formulated as an MINLP, linearized, and reworked into a two-phase heuristic to minimize resource costs. Video streaming services can benefit from low latency of fog, but creates a carbon footprint as a result [4]. In response, Do et al. [4] formulate the resource allocation of these services as a convex optimization to minimize carbon footprint. They then develop an iterative heuristic inspired by proximal algorithms and Alternating Direction Method of Multipliers to serve video streaming services with guaranteed bandwidth and low carbon footprint. The *Mist* [17] crowd-sensing infrastructure also reserves VMs in fog for IoT use; the placement of VMs in fog are formulated as a linearized MINLP, and solved to allocate IoT with minimal VM deployment cost.

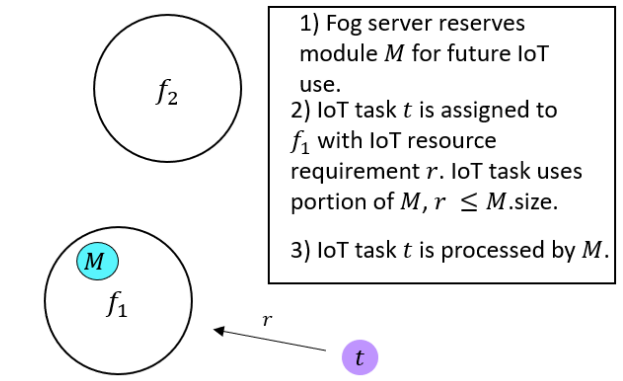
VI. Fog Computing Frameworks

Fog orchestration is defined by using a *control* layer to periodically monitor the available resources and request allocations to each fog node [99, 100]. Instead of an IoT request being uploaded directly to the fog layer, it is uploaded to the control layer which then distributes the application to fog nodes or Cloud accordingly. *Fog frameworks* can be seen as specific applications of fog orchestration to facilitate resource provisioning & allocation via a *fog orchestration controller* (FOC) or an API; the basic functionality of fog frameworks is shown in Fig. 6. If a module migration is required between fog nodes under the same FOC, the sending fog node will either consult the FOC to determine where to send the module, or send the module to the FOC for re-distribution. The fog layer may be partitioned into groups of fog nodes, each group with a separate FOC. In this case, a module migrated outside a cluster is sent to the FOC, which communicates with other FOCs to determine the module destination. If an API model is used for migration, a fog node will query other fog nodes to determine where to migrate data.

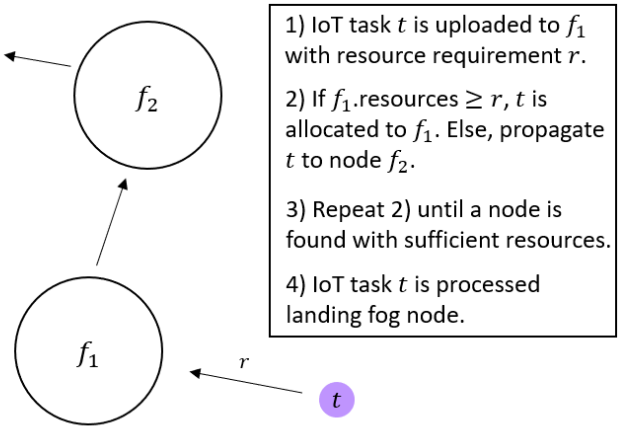
The frameworks covered in this section provide resource management of fog nodes via hardware and/or software extensions. By monitoring fog resource availability, a framework can distribute IoT requests to the fog without fog rejection, thus minimizing latency. In addition, certain frameworks allow data migration between fog nodes when an initial fog node has insufficient resources.

A. Non-fixed fog topologies

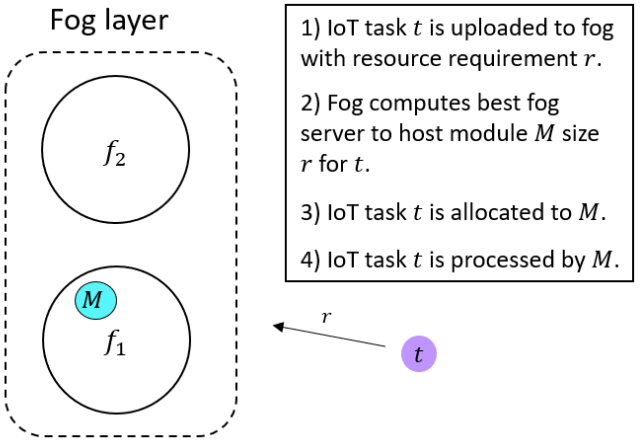
Chen et al. [38] define *fog-as-a-service-technology* (FA^{2}ST) as a fog framework that can support any IoT application, i.e. regardless of use case, with the objective of providing value-added fog services compared to Cloud. Importantly, FA^{2}ST provides on-demand fog service discovery [101], allowing it to probe all connected fog nodes for current resource availability at the moment an IoT request arrives. Furthermore, if an IoT application is assigned to a fog node that is no longer available, FA^{2}ST re-deploys the application to a new node. This allows FA^{2}ST to service a fog infrastructure with faulty, mobile, or dynamically available fog nodes. Madan et al. [102] envisions an IoV-fog infrastructure that provides UAVs to support overloaded RSUs. From a base station where all UAVs are kept and charged, an overloaded RSU triggers a UAV deployment to travel and hover over the RSU; data is migrated



(a) Prior provisioning and prompt allocation.

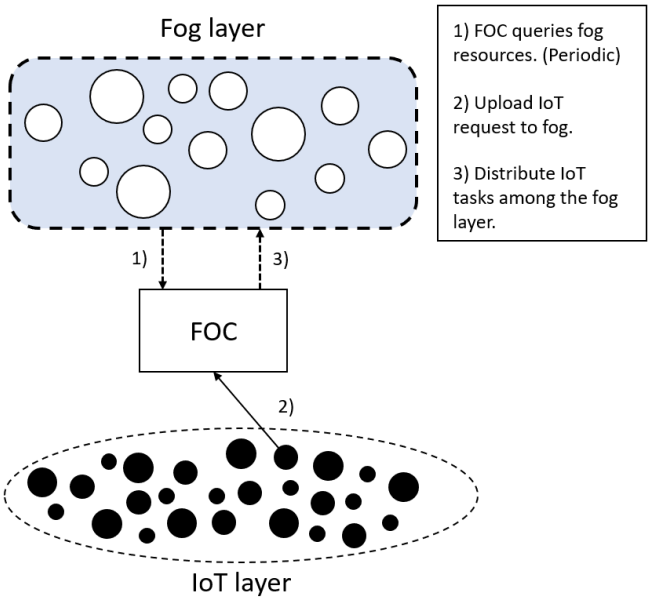


(b) On-demand provisioning and prompt allocation.

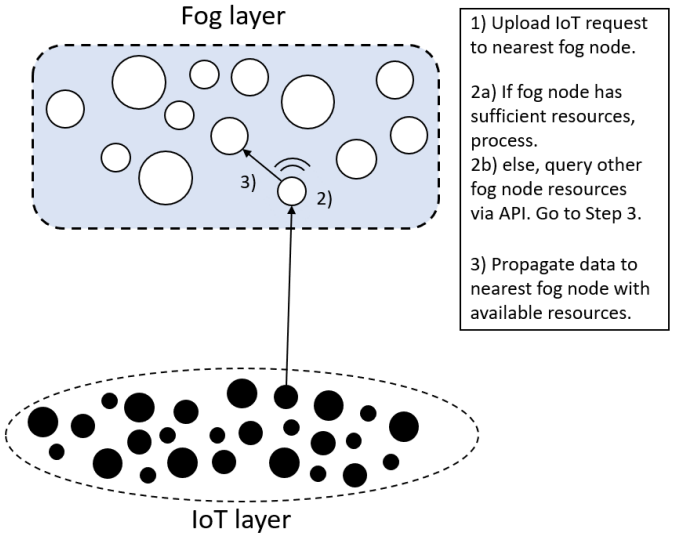


(c) On-demand provisioning and small batch allocation.

Figure 5: Resource management methods using a) prior provisioning and prompt allocation schemes, where fog resources are reserved for predicted IoT use, b) on-demand provisioning and prompt allocation schemes, where IoT task searches for suitable node, and c) on-demand provisioning and small batch allocation schemes, where optimal placement is found prior to processing. In b) and c), secondary node f_2 may be either a fog node or Cloud.



(a) FOC method.



(b) API method.

Figure 6: A fog framework provides current resource availability information of all fog nodes to streamline the resource allocation process. Two standard approaches are to use an FOC (hardware overhead) or to use an API (software overhead) to query fog and distribute IoT tasks.

to the UAV to accelerate the IoV computation and decrease response latency.

A framework can have a single FOC with a ubiquitous view of all fog resources [16, 98], or multiple FOCs, each monitoring a cluster of fog nodes [9, 29, 35]. A single FOC requires only one additional node; however it must have a connection to every fog node which may not be scalable for larger fog layers. Multiple FOCs provide improved scalability at a cost of increased hardware overhead. Furthermore, a module migration between FOCs may result in increased latency.

B. Fixed topologies

For a fixed fog topology, frameworks optimize the communication between end devices, fog, and Cloud. This approach to having an ubiquitous view of all fog information is through the use of an FOC which is connected to every fog either directly [9, 16, 43] or through APIs [17, 37, 39, 43, 56, 103, 104]. In addition to resource management for efficient IoT-fog service provisioning, most fog frameworks claim to be scalable to increased IoT traffic volume or fog infrastructure size [29, 35, 39, 43, 98].

Cardellini et al. [93] provide a distributed extension to the open source data stream processing application *Storm* to allow the execution of a distributed QoS-aware IoT resource scheduler. This extension allows fog nodes within the infrastructure to have knowledge of other fog node resource availability, and for IoT service scheduling to respect latency and resource requirements.

Donassolo et al. [98] propose *Fog-IoT ORchestator* (FITOR), a fog framework that monitors the fog infrastructure to survey and extract resource metrics from all fog nodes. This allows FITOR to have a ubiquitous view of all fog resources at any time, facilitating the service deployment of IoT data to fog nodes. As a result, FITOR is used in conjunction with the O-FSP resource provisioning scheme [98]. In order to allow fog nodes to accept any IoT request, FITOR must receive service descriptors from IoT applications along with the request. These descriptors define the IoT application, its building components and QoS requirements. Yigitoglu et al. [16] introduce the *Foggy* framework which accepts similar information from IoT, and deploys each task of an IoT request to a fog node with available resources and satisfying latency, privacy and priority requirements. The Foggy FOC monitors all fog resources via the MQTT protocol [105], and stores historical IoT requirements and workloads for faster future deployment planning. Although these frameworks aim to satisfy QoS of IoT requests by querying the requests, expecting requirement information from IoT may not be feasible.

Tuli et al. [35] propose *FogBus*, a scalable fog framework that partitions fog nodes into separate roles to increase security and reliability. FogBus is composed of fog gateway nodes, fog broker nodes (FOC), general computing nodes, and fog repository nodes. IoT applications upload data directly to fog gateway nodes, which are then forwarded to interconnected fog broker nodes, and distributed to fog general computing nodes for IoT request processing. Each fog broker node is also connected to a fog repository node which facilitates data sharing, replication, recovery and secured storage. Replication of stored data across multiple fog nodes provides storage robustness and reliability for possible fog node failures. Indeed, data stored on faulty fog brokers can be recovered by fog repository nodes, and replaced by existing fog broker nodes. FogBus also supports blockchain integration to verify that IoT data is coming from a pre-defined credible source.

Skarlat et al. [9] propose a fog architecture with groups of fog nodes clustered into *colonies* with all fog resource information available to an FOC. Upon receiving an IoT request, the FOC allocates IoT tasks among the fog nodes in the colony. If a fog colony does not have enough resources to process the IoT request, the FOC can query other colonies for available resources and transfer the request accordingly via REST APIs. It is recognized that not all IoT requests are suited for fog [9], e.g., non-delay sensitive requests or resource intensive big data. As a result, [9] provides Cloud-fog middleware to propagate these applications for Cloud resource allocation.

Zhang et al. [29] consider data migration between RSUs and fog nodes of their proposed IoV-fog application. Several fog nodes are clustered to provide seamless data sharing between nodes in a select region. Between clusters exists a *coordinator* (FOC) that manages fog resources over the system. If a fog cluster requires extra resources, or observes a vehicle moving towards a new region, it may handover an IoT module to a neighbouring fog cluster to continue uninterrupted IoT processing.

C. SDN-based frameworks

An SDN decouples the control plane and data planes of a traditional network, allowing an SDN controller to forward data through the network based on an arbitrary rule set [106]. Combining SDN with a standardized switch like OpenFlow [64] provides a simpler interface for SDN controllers to interact with the network. Furthermore, SDN can track mobile IoT devices to predict future destinations and support seamless data handover between fog nodes for uninterrupted IoT support [39].

Tomovic et al. [39] propose to achieve resource management, traffic control, and data migration of the fog plane via an SDN with OpenFlow controllers [64] which supports API functionality for IoT deployment. The Mist architectural framework [17] similarly uses SDN to monitor fog infrastructure resources, and APIs to monitor both IoT and fog device health, and facilitate IoT service allocation to fog.

Yousefpour et al. [43] propose *FogPlan*, a lightweight QoS-aware framework that uses an SDN controller to monitor incoming IoT traffic and fog node resources, and deploy IoT processing accordingly via an API. Notably, FogPlan aims to satisfy IoT QoS requirements with no or minimal IoT requirement information, increasing the workload of FogPlan while decreasing that of IoT devices. They use FogPlan to enable the min-cost and min-viol real-time resource provisioning scheme [43].

D. Data Migration

The *Foglet* [37] and *Mobile Fog* [103] programming models focus on module migration between IoT devices and fog nodes in a hierarchical architecture. They both implement an API that allows modules to either migrate one hierarchical level above or below, or to move to a specific fog node.

The data migration implementation with foglets [37] supports IoT requests that can simultaneously use different parts of the fog-Cloud infrastructure for different IoT tasks. Mobile Fog [103] creates a dynamic scaling mechanism whereby new on-demand fog nodes are created in response to overloaded workloads, and data is appropriately distributed to new fog

nodes. Dynamic scaling with API data migration support creates a programming model well suited for mobile IoT devices such as smart cars, smart phones, and smart cameras.

Zhou et al. [47] propose a *vehicular fog computing* (VFC) framework with associated base stations to provide intra-fog resource management. Since a data user uploads their data to the nearest fog node, it is possible for a node to become overloaded during peak usage periods. In these cases, VFC allows for task offloading to nearby underutilized smart vehicles.

VII. Fog Infrastructure Design Evaluation

The efficiency or desirability of a fog computing infrastructure can be evaluated using simulation and emulation. Furthermore, simulating/emulating an existing fog infrastructure can be useful to fog application developers in deciding whether application processing should be done locally, by fog, or by Cloud [107]. There exists a variety of tools that deploy custom resource management policies, each focusing on different evaluation metrics and use case support. In the implementation of fog evaluation tools, the allocation and migration of modules are considered in terms of VMs. For more complex dynamic fog infrastructures, emulation frameworks can evaluate service of IoT applications in a way that more closely mimics the real fog infrastructure.

A. Simulation

In all simulation cases, a fog infrastructure is first defined manually by the user. Based on simulated IoT traffic parameters, the effects of network flow through fog can be evaluated. In most cases, the routing and interactions with fog nodes are defined by fog processing and forwarding policies. In the case of FogExplorer [108, 109], each IoT request is mapped to a selected machine for processing, and the effects of the deployment mapping on cost and QoS are evaluated. Each IoT request can be mapped to self, a specific fog node, or to Cloud. While designing an IoT application, testing different mapping options enables developers to identify the best location for IoT application processing [107].

iFogSim [36], an extension of CloudSim [95], is a simulation toolkit for IoT and fog that allows users to measure the resource cost, network use, energy consumption and latency of a specific network and resource management policy. Many resource management schemes use iFogSim to measure the impact of their proposed contributions [41, 34, 56, 110, 111]. EdgeNetworkCloudSim [112, 113] is also an extension of CloudSim that allows the simulation and evaluation of user-defined algorithms for placement, orchestration and consolidation of service chains. The system frequently monitors fog resources and energy consumption, and logs IoT task rejections due to insufficient fog resources. It does not however take into consideration the latency of IoT allocations.

Multiple tools extend iFogSim to address scenarios that cannot be simulated with iFogSim alone. iFogSimWithDataPlacement [110, 114] allows the implementation of resource management strategies that optimize data placement for a selected metric; this is achieved through MILP, and divide & conquer algorithm support. FogWorkflowSim [115] combines elements from WorkflowSim [116] and iFogSim to model and simulate workflows in fog. Identifying tasks that can be processed in parallel within a workflow decreases the overall latency and energy consumption [115]. Both MyiFogSim [117, 118] and MobFogSim [119] allow users to define the migration of VMs allocated to a mobile IoT device. The migration policy determines when a VM should be migrated by defining a migration zone around a Fog node, and a migration point an IoT device must cross to signal that it is moving away from the Fog node. The migration strategy defines where and how to migrate the VM. To where a VM should be migrated is dictated by the speed and direction of mobility, while the migration itself uses one of many strategies with different possible VM down time and memory usage during transfer. This scenario is shown in Fig. 7a.

FogNetSim++ [120, 121] extends fog support to the OMNet++ [122] discrete event simulator. The system uses a fog broker node to monitor fog resources, and to allocate IoT requests using a greedy approach by distance. Uniquely, FogNetSim++ allows for mobile fog node modelling. If a fog node moves away from an allocated IoT device, the resource module is migrated to the nearest fog node within the IoT device's range. This scenario is shown in Fig. 7b.

YAFS [123] is a discrete-event simulator that provides highly customizable placement, scheduling and routing strategies for IoT requests in fog. YAFS considers fog server failures, modeled using an exponential distribution. PureEdgeSim [124, 125] defines a total energy attribute for each fog server, along with energy consumption per processed IoT task. A fog server fails when its total energy is depleted. FogDirSim [126, 127] is a tool to simulate CISCO FogDirector, an IoT/fog manager. The simulator probabilistically predicts fog utilization, energy consumption, and robustness to fog node failures of a user-defined resource management policy. For these three simulators [123, 124, 126], a fog server failure triggers the re-allocation of modules present in the failed node are re-allocated. This scenario is shown in Fig. 7c.

PFogSim [91] defines a multi-tier fog architecture with Cloud in the final layer. The custom resource management policies define which layers can participate in IoT processing, and how IoT processes are distributed among the valid layers to optimize latency or operational cost. By restricting the scope of IoT assignment, a user can test various infrastructures to determine a minimal viable infrastructure. Therefore, PFogSim may serve as an empirical method for fog design using a multi-tier architecture.

Most evaluation tools use discrete-event simulation to simulate and evaluate the life-cycle of a fog infrastructure. In contrast, FogTorchΠ [128] uses Monte Carlo simulations to generate several possible IoT assignments that satisfy IoT QoS, hardware and software requirements. This approach allows a user to test resource allocation feasibility under several circumstances and requirements. The latency and bandwidth requirements from each simulation is analyzed to determine the set of resource deployments that yield the best estimated QoS.

B. Emulation

Simulation is a cost effective and computation efficient method of evaluating IoT interactions with a fog infrastructure; however, it usually assumes a number of simplifications that may not mimic a dynamic fog infrastructure well. In complex systems, *emulation* duplicates the fog topology and IoT workload, providing repeatable and controllable experiments of real IoT applications [129].

Héctor [130], MockFog [131, 132], and EmuFog [129, 133] are three fog emulation toolkits that aim to provide more realistic testing of fog infrastructures and IoT applications. In addition to emulating the fog infrastructure, Héctor also represents each IoT device as a VM with configurable properties, creating one emulated IoT environment. A *testbed* is a set of configured IoT request patterns, IoT resource requirements, and fog network conditions such as packet loss and random additional delay. Automated execution of various testbeds can therefore mimic the most realistic flow of IoT-fog interactions. MockFog allows users to inject failures into the fog system by toggling select fog nodes as unavailable. This will reduce fog resource availability in order to further test fog infrastructure resiliency and fault-tolerance. These three emulators allow users to fully define and design a fog infrastructure from scratch, though EmuFog also allows users to place fog nodes along the topology using a default greedy approach, or a custom placement policy. By changing the fog design, placement policies, or network conditions, users can evaluate the effects of the different fog infrastructures on IoT service. In this manner, emulation toolkits may serve as an empirical method for fog design.

VIII. Open Issues & Research Opportunities

A. Fog Design & Dimensioning

Both [5] and [78] are based on data traffic from static IoT devices; they assume a reliable fog infrastructure. In practice, many IoT devices are dynamic in location and request frequency, leading to possible spikes in incoming IoT traffic. Likewise, fog node failures will lead to spikes in incoming IoT traffic to active fog nodes. Updated simulations of fluctuating network congestion on the edge and stochastic fog node failures can serve useful to understanding the latency impact from fog. A fog design & dimensioning solution can be sensitive to network traffic, fog reliability, and fog modeling parameters such as candidate resource location and quantity. Therefore, a fog design & dimensioning solution should provide satisfactory QoS conditions to IoT under volatile IoT traffic and fog node failures, up to a degree of confidence.

In the event of a node failure, all IoT requests on the failed fog node should be re-allocated. One approach to ensure successful re-allocation is to implement the addition of repository nodes in the design of a fog infrastructure to backup fog node data. Adding mechanisms for IoT re-allocation will increase the fault-tolerance of the designed fog infrastructure.

The extensibility of [78] assumes more fog nodes are added for a static increase in IoT requests; however, if the added fog nodes increase the geographic range of the fog infrastructure, then previously out-of-range IoT devices may now be in range. The resulting total IoT traffic to the extended fog infrastructure may increase to more than estimated. IoT traffic patterns may evolve over time, and IoT traffic may reduce in certain areas and increase in others. An extension of current fog infrastructure should identify fog resources that can be repurposed to other fog nodes to optimize extension costs. Simulation of increased IoT traffic from geographic extensions to the fog infrastructure can be useful to understand fog extensibility beyond what is currently proposed.

These open issues provide an opportunity for future research, including the use of dynamically located and available fog servers, and fault-tolerance.

B. Fog Resource Provisioning & IoT Resource Allocation

In many cases, it is assumed that fog resources are previously known to IoT devices [4, 29, 43, 79, 85, 97] and often provided by an FOC with connections to every fog node [9, 98]. Some schemes assume a fog server in range of IoT satisfies latency [82, 98], or that current fog infrastructure follows certain architectural characteristics [14, 29, 89].

The standard approach with prompt allocation schemes is to upload IoT data to the nearest fog server, regardless of available fog resources or fog resource pricing. Indeed, it is assumed that this information is not known prior to IoT-fog connection. This can result in sub-optimal processing costs in a multi-price fog environment. If the nearest fog server is heavily congested with requests, the response latency can increase depending on whether the IoT request waits in a queue for processing [56], is propagated to different fog nodes [29], or is propagated to Cloud [30, 56]. This is shown in Fig. 5b. Although prior provisioning approaches [14, 81] help mitigate IoT requests being re-distributed elsewhere, resource cost concerns persist. As shown in Fig. 5a, a set of resources are reserved per fog node based on IoT traffic predictions. The fog node immediately allocates an incoming IoT request to the reserved resources, reducing latency; however, this assumes that reserved resources are always sufficient for IoT use. Another approach to mitigate re-distribution is to have a system-wide knowledge of fog resources [29, 43], however this generates additional hardware and/or software overhead.

The machine learning approach with prompt allocation schemes is to use RL to improve the efficiency of fog resource management over time [87, 88, 94]. However, QoS requirements of IoT requests are assumed to be met through the use of fog, and not verified. Currently, all current RL applications for fog resource management use a centralized RL agent. For large systems, the use of distributed multi-agent RL, each overseeing a fraction of total fog nodes, may improve system resource management as a whole. To reduce computation overhead, each RL agent should learn to independently behave optimally without requiring cooperation between them. Most resource management schemes assign a single IoT request to multiple modules in separate fog nodes without consideration of fog routing costs. Multiple multi-agent RL applications have been successful in decreasing network latency [134, 135], energy consumption [136] or cost [137] of general network routing between a source and

target node. There has yet to be a distributed RL application for fog. Therefore, research opportunities exist for fog resource management using distributed RL for module assignment and inter-module routing.

Schemes that make an effort to provide QoS satisfying allocations generate small amounts of latency overhead [85] which may prove too large for certain IoT applications, or are currently only effective for IoV applications [84]. Future research is required into prompt allocation that does not jeopardize latency in the worst-case, and provides near-optimal resource allocation costs with little overhead. The objectives and modeling techniques of reviewed prompt resource allocation schemes are summarized in Table IV.

Small batch allocation schemes have longer resource provisioning & allocation solution times for better cost or latency, making them ideal for static IoT devices needing long-term and frequent fog support; however, mobile single-request IoT devices may also request resources from fog. Further research is required to provide resource provisioning & allocation for both static and dynamic IoT devices.

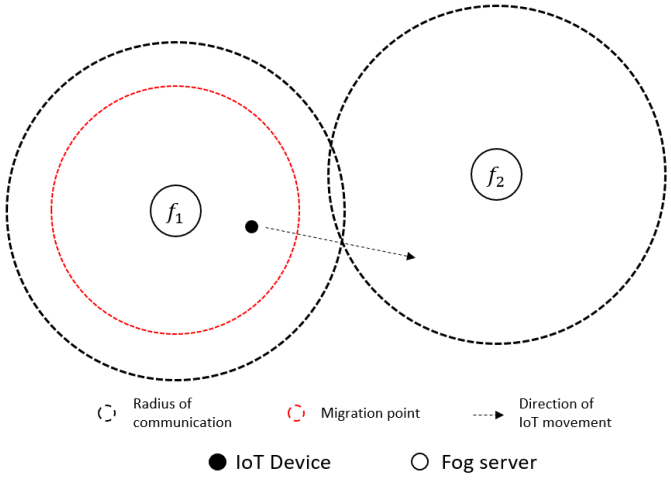
All small batch allocation schemes are determined to assume: 1) a static set of IoT requests for the duration of allocation optimization, 2) guaranteed bandwidth availability, and 3) all current fog resources are known. Each model is formulated to service IoT requests in batches, which may require additional latency for a sufficiently large batch to accumulate. The objectives and modeling techniques of reviewed small batch allocation schemes are summarized in Table V.

C. Fog Computing Frameworks

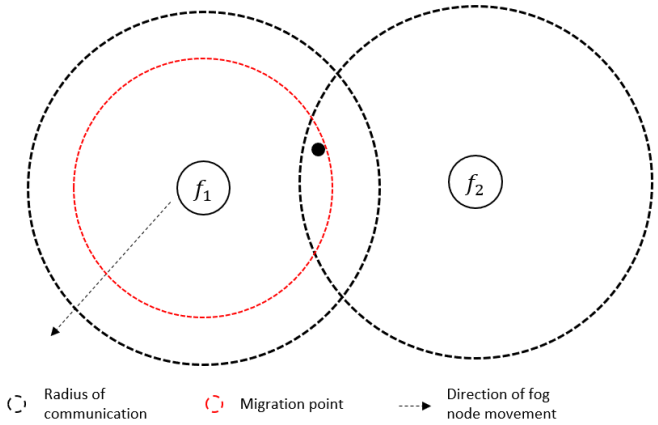
An ideal fog framework can monitor resources from the fog layer, and provide possible data migration between fog servers with minimal additional latency and overhead. Additional hardware must be able to communicate efficiently with existing fog infrastructure, which may be problematic if the framework and infrastructure are owned by different entities. Similarly, additional software such as API support must be installed on the individual fog nodes, implying the framework owner has access to the fog node software. An ideal FOC requires no additional information from IoT regarding the latency, privacy or security QoS, and can support any heterogeneous IoT request.

A single fog node provides privacy when data is stored locally; however, using an FOC that distributes an IoT request among a cluster of fog nodes can broaden the scope of data exposure to multiple fog nodes. This creates a trade-off between larger clusters of fog nodes under one FOC which provides allocation efficiency, and increased data privacy when using smaller clusters.

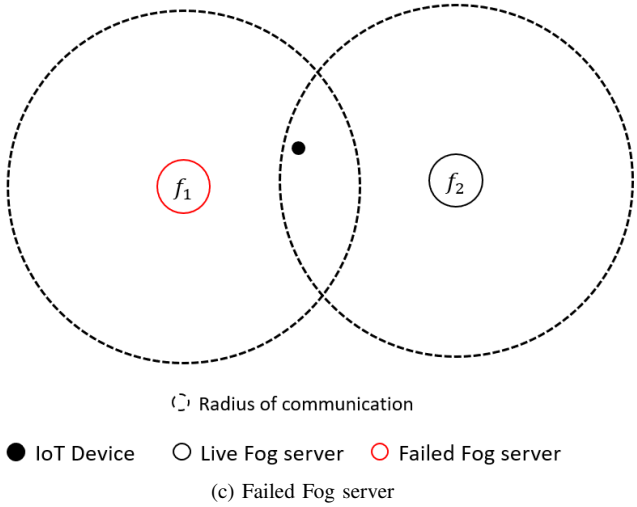
To the best of our knowledge, all mentioned fog frameworks assume ownership of the fog infrastructure, simplifying all interactions between the fog framework and the fog infrastructure. In a scenario with multiple public fog servers with different owners, a framework for each owner could create redundant overhead. Further research is needed to propose a fog framework that systematically addresses issues of generated overhead, fog infrastructure/framework ownership,



(a) Mobile IoT device



(b) Mobile Fog server



(c) Failed Fog server

Figure 7: Module migration due to: a) mobile IoT device moving between fog server ranges, b) mobile fog server moving out of IoT device range, and c) fog server failure. In all scenarios, the IoT task is initially connected to f_1, and is migrated to f_2. In a) and b), the IoT task begins migration once the IoT device crosses the migration point.

Table IV: Summary of resource provisioning schemes with prompt allocation.

Year	Author	Optimization objective	Modeling techniques
2015	Aazam and Huh [14, 81]	Fair resource pricing and prior provisioning based on historical IoT relinquish behaviour.	Formula to determine quantity of resources to allocate to IoT.
2016	Aazam et al. [82]	Extends prior provisioning of [14, 81] to factor QoE.	Formula to determine quantity of resource to allocate to IoT.
2016	Agarwal et al. [30]	Maximize fog utilization.	Efficient resource allocation algorithm.
2017	Zhang et al. [29]	Minimize energy-consumption of intra-fog resource management, minimize IoT service rejection of inter-fog resource management.	Convex optimization, and min-max optimization.
2017	Bittencourt et al. [56]	Minimize latency.	Evaluation of three IoT processing policies – concurrent, FIFO, delay-priority.
2018	Yousefpour et al. [43]	Minimize cost or delay violations.	Greedy algorithms with periodic execution.
2018	Xia et al. [85]	Minimize latency.	Anchor-based fog node ordering and dynamic component ordering heuristics.
2018	Wei et al. [87]	Minimize latency.	Reinforcement learning.
2018	Sun et al. [94]	Minimize power consumption of fog.	Reinforcement learning.
2018	Wang et al. [88]	Minimize latency.	Reinforcement learning.
2019	Aazam et al. [83]	Predicts and prior provisions resource consumption based on QoS and QoE.	Formula to determine quantity of resource to allocate to IoT.
2019	Zhou et al. [47]	Maximize vehicular fog utility.	Base station conducted pricing-based stable matching algorithm.
2020	Peng et al. [84]	Match vehicular fog nodes with IoT device based on resource, latency, reputation and pricing requirements of both parties.	One-to-one assignment algorithm to match vehicular fog nodes with clients.

Table V: Summary of resource provisioning schemes with small batch allocation.

Year	Author	Optimization objective	Modeling techniques
2015	Gu et al. [28]	Minimize cost.	MILP-based two-phase heuristic.
2015	Do et al. [4]	Minimize carbon footprint.	Distributed algorithm based on proximal algorithm and alternating direction method of multipliers.
2016	Zeng et al. [89]	Minimze latency.	MILP-based three-stage heuristics.
2016	Souza et al. [32]	Minimize cost.	MILP.
2016	Zhang et al. [90]	Minimize cost.	Stackelberg game between Fog nodes, MDCOs and IoT.
2016	Intharawijitr et al. [31]	Minimize blocking probability.	Evaluation of three IoT assignment policies — random, lowest latency fog node, maximum resource capacity Fog.
2016, 2017	Skarlat et al. [9, 138]	Maximize fog utilization.	MILP and genetic algorithm.
2017	Arkian et al. [17]	Minimize cost.	Linearized MINLP.
2017	Taneja and Davy [79]	Optimize latency, fog utilization and energy consumption.	ModuleMapping algorithm.
2017	Ni et al. [96]	1) Minimize latency, 2) maximize fog utility.	Heuristic algorithm.
2018	Ali et al. [92]	Minimize latency.	Many-to-one matching algorithm.
2019	Donassolo et al [98]	Minimize cost.	MILP model for exact solution, heuristic algorithm for real-time allocation.
2019	Salaht et al. [97]	Satisfy QoS.	Constraint Programming.
2019	Karamoozian et al. [80]	Minimize latency.	Gravitational Search Algorithm.

Table VI: Summary of contributions with fog frameworks and data migration.

Year	Author	Main Contribution	Overhead	
			FOC	API
2013	Hong et al. [103]	*Mobile fog* programming model for data migration between IoT and fog nodes.	No	Yes
2015	Gu et al. [28]	Fog infrastructure for medical cyber-physical systems with possible data migration among fog nodes.	Multiple	No
2015	Cardellini et al. [93]	Frequent resource availability sharing between fog nodes.	Single	No
2016	Saurez et al. [37]	*Foglet* programming model for data migration between IoT and fog nodes.	No	Yes
2016	Skarlat et al. [9, 138]	Groups fog nodes into colonies, each with a data migration controller to transfer IoT data between colonies if current fog colony resources are insufficient.	Multiple	No
2017	Yigitoglu et al. [16]	*Foggy* fog framework allows ubiquitous view of all fog resources, deploys IoT to fog nodes satisfying latency, privacy and priority requirements.	Single	No
2017	Zhang et al. [29]	Data migration between RSUs and fog nodes.	Multiple	No
2017	Tomovic et al. [39]	Resource management, traffic control and data migration via SDN.	No	Yes
2018	Yousefpour et al. [43]	*FogPlan* lightweight QoS-aware framework using SDN to control and monitor incoming IoT traffic and fog resources.	No	Yes
2018	Chen et al. [38]	FA^2ST, infrastructure of cross-domain supporting fog nodes.	Single	No
2019	Zhou et al. [47]	Vehicular fog framework with intra-fog resource management. If a fog becomes overloaded, allows task offloading to nearby smart vehicles.	Multiple	No
2019	Tuli et al. [35]	*FogBus* fog framework partitions fog nodes into separate roles and IoT exposure to increase fog security and reliability.	Multiple	No
2019	Donassolo et al. [98]	*FITOR*, IoT-fog orchestration framework that allows ubiquitous view of all fog resources.	Single	No
2020	Madan et al. [102]	Allows overloaded RSUs to call and offload data to UAVs.	Multiple	No
2020	Peng et al. [84]	Multi-attribute double auction vehicular framework to match and pair vehicular fog nodes with IoT devices based on resources, latency, reputation and pricing requirements of both parties.	Multiple	No

Table VII: Summary of discrete-event simulation and emulation toolkits for Fog computing infrastructure — Functionality

Year	Name	Method	Migration of IoT process due to			Topology input
			Mobile IoT	Fog node failure	Mobile Fog node	
2017	EmuFog [129, 133]	Emulation	No	No	No	BRITE [139]
2017	iFogSim [36, 140]	Simulation	No	No	No	GUI, JSON
2017	MyiFogSim [117, 118]	Simulation	Yes	No	No	GUI, JSON
2017	EdgeNetworkCloudSim [112, 113]	Simulation	No	No	No	BRITE [139]
2018	FogExplorer [108, 109]	Simulation	No	No	No	GUI
2018	iFogSimWithDataPlacement [110, 114]	Simulation	No	No	No	GUI, JSON
2018	FogNetSim++ [120, 121]	Simulation	No	No	Yes	.INI file
2019	PFogSim [91]	Simulation	No	No	No	XML
2019	YAFS [123]	Simulation	No	Yes	No	JSON
2019	PureEdgeSim [124]	Simulation	No	Yes	No	Java
2019	FogWorkflowSim [115]	Simulation	No	No	No	Java
2019	Héctor [130]	Emulation	No	No	No	API
2019	MockFog [131, 132]	Emulation	No	No	No	API, GUI
2020	FogDirSim [126, 127]	Simulation	No	Yes	No	Database
2020	MobFogSim [119]	Simulation	Yes	No	No	GUI, JSON

Table VIII: Summary of discrete-event simulation and emulation toolkits for Fog computing infrastructure – Evaluation Criteria

Year	Name	Method	Evaluated Metrics				
			Latency	Operational Costs	Energy consumption	Fog utilization/ congestion	Task Rejection
2017	EmuFog [129, 133]	Emulation	Yes	No	Yes	Yes	No
2017	iFogSim [36, 140]	Simulation	Yes	Yes	Yes	Yes	No
2017	MyiFogSim [117, 118]	Simulation	Yes	Yes	Yes	Yes	No
2017	EdgeNetworkCloudSim [112, 113]	Simulation	No	No	Yes	Yes	Yes
2018	FogExplorer [108, 109]	Simulation	Yes	Yes	No	No	No
2018	iFogSimWithDataPlacement [110, 114]	Simulation	Yes	Yes	Yes	Yes	No
2018	FogNetSim++ [120, 121]	Simulation	Yes	No	Yes	No	Yes
2019	PFogSim [91]	Simulation	Yes	Yes	No	No	Yes
2019	YAFS [123]	Simulation	Yes	No	No	Yes	Yes
2019	PureEdgeSim [124]	Simulation	Yes	No	Yes	Yes	No
2019	FogWorkflowSim [115]	Simulation	Yes	Yes	Yes	No	No
2019	Héctor [130]	Emulation	Yes	No	No	Yes	No
2019	MockFog [131, 132]	Emulation	Yes	No	No	Yes	Yes
2020	FogDirSim [126, 127]	Simulation	No	No	Yes	No	Yes
2020	MobFogSim [119]	Simulation	Yes	Yes	Yes	Yes	No

privacy, and communication with IoT applications. There is also a research opportunity to develop a proof-of-concept of a framework that can interact with existing public fog infrastructure regardless of owner. The main contributions and hardware/software overhead generated for each fog framework is summarized in Table VI.

D. Fog Infrastructure Design Evaluation

After an IoT request is assigned to a fog node, there are several scenarios in which the provisioned module needs to be migrated to a new fog node. Specifically, we consider module migration due to a mobile IoT device moving between fog nodes, a mobile fog node moving away from an IoT device, and a fog node failure. When a mobile IoT moves between fog nodes, module migration is efficient for slow moving devices; however, significant decreases in successful module migrations is seen with fast vehicles [119]. The proposed fog implementations with mobile fog nodes use slow moving UAVs [141] and buses [46], ensuring migration efficiency. Though module handover is supported in FogNetSim++ [120], specifics regarding migration policies and strategies are not fully explored beyond what is available in OMNet++ [122]. When fog node failures are present, it is necessary for replicas of the module data to be stored elsewhere in order to restore and migrate data to live fog nodes [35].

For all current simulators, at most one of these scenarios are explicitly addressed. Though fog computing has many use cases that require module migration, there is yet to be a tool that simulates all possible scenarios. For each described evaluation tool, the approach to simulation/emulation is only appropriate if the designed fog infrastructure and framework can perform similar actions.

From an implementation standpoint, file input support is ideal in situations where the entities designing the fog infrastructure and evaluating its efficiency are different. Using formats like JSON, XML or BRITE [139] can facilitate the fog development process across multiple entities in an organization. A GUI or topology input directly into the evaluation tool can be useful when trial-and-error is required to find an optimal design. We wish to mitigate this using a fog design & dimensioning process [5, 78]. The supported migration scenarios and topological input format of each simulator is summarized in Table VII.

Each tool records logs during evaluation, outlining metrics of each task allocation such as latency, energy consumption, cost of allocation, resulting network congestion, and task rejection due to unsatisfied QoS requirements. Though most tools allow custom output metrics, only what is recorded in logs can be evaluated; indeed, no evaluation tool records all evaluation metrics shown in Table VIII.

There remain opportunities to build an evaluation tool with the flexibility to model any scenario in a fog infrastructure. This becomes particularly important in a dynamic fog system with fog node failures, dynamically available fog nodes, mobile fog nodes, and mobile IoT. Since the evaluated output can only aggregate what is logged, there is also an opportunity to expand the available metrics that are recorded from the IoT request, the provisioned fog node, and the network path.

IX. Conclusion

Fog computing provides an alternative to Cloud processing with increased privacy and low latency to IoT applications. As the number of IoT applications increases, fog infrastructure implementations become crucial. We have identified four phases to implement a fog infrastructure, and discussed their limitations and open issues. We compared current design & dimensioning models which produce a blueprint of a fog infrastructure for the support of IoT applications. We classified resource provisioning & allocation schemes by their effectiveness to support dynamic and static IoT applications, and identified optimal objectives and modelling techniques used by each scheme. We analyzed the main contributions and generated overhead of fog frameworks. Finally, we reviewed and identify limitations of simulation/emulation tools for the evaluation of the designed fog infrastructure.

With this survey of current fog solutions, we intend to give a detailed understanding of the necessary considerations of building a practical fog infrastructure for local IoT support, up to large-scale Smart City systems. Based on our results, we believe we have provided a detailed overview of steps necessary for a practical and functional fog implementation.

References

[1] IDC. *The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. Tech. rep. Accessed: 2020-04. 2019. URL: https://www.idc.com/getdoc.jsp?containerId=prUS45213219.

[2] CISCO. *Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are*. Tech. rep. Accessed: 2019-04. 2015. URL: https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf.

[3] G. L. Santos, P. T. Endo, M. F. F. da Silva Lisboa, L. G. F. da Silva, D. Sadok, J. Kelner, T. Lynn, et al. "Analyzing the availability and performance of an e-health system integrated with edge, fog and cloud infrastructures". In: *Journal of Cloud Computing* 7.1 (2018), p. 16.

[4] C. T. Do, N. H. Tran, C. Pham, M. G. R. Alam, J. H. Son, and C. S. Hong. "A proximal algorithm for joint resource allocation and minimizing carbon footprint in geo-distributed fog computing". In: *2015 International Conference on Information Networking (ICOIN)*. IEEE. 2015, pp. 324–329.

[5] C. Yu, B. Lin, P. Guo, W. Zhang, S. Li, and R. He. "Deployment and Dimensioning of Fog Computing-Based Internet of Vehicle Infrastructure for Autonomous Driving". In: *IEEE Internet of Things Journal* 6.1 (2019), pp. 149–160.

[6] S. W. Loke. "The internet of flying-things: Opportunities and challenges with airborne fog computing and mobile cloud in the clouds". In: *arXiv preprint arXiv:1507.04492* (2015).

[7] L. M. Vaquero and L. Rodero-Merino. "Finding your way in the fog: Towards a comprehensive definition of fog computing". In: *ACM SIGCOMM Computer Communication Review* 44.5 (2014), pp. 27–32.

[8] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. "Fog computing and its role in the internet of things". In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM. 2012, pp. 13–16.

[9] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner. "Resource provisioning for IoT services in the fog". In: *2016 IEEE 9th international conference on service-oriented computing and applications (SOCA)*. IEEE. 2016, pp. 32–39.

[10] A. Giordano, G. Spezzano, and A. Vinci. "Smart agents and fog computing for smart city applications". In: *International Conference on Smart Cities*. Springer. 2016, pp. 137–146.

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

[12] A. Ahmed, H. Arkian, D. Battulga, A. J. Fahs, M. Farhadi, D. Giouroukis, A. Gougeon, F. O. Gutierrez, G. Pierre, P. R. Souza Jr, et al. "Fog Computing Applications: Taxonomy and Requirements". In: *arXiv preprint arXiv:1907.11621* (2019).

[13] K. Bachmann. "Design and implementation of a fog computing framework". Diploma Thesis. Vienna University of Technology (TU Wien), Vienna, Austria, 2017.

[14] M. Aazam and E.-N. Huh. "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT". In: *2015 IEEE 29th International*

Conference on Advanced Information Networking and Applications. IEEE. 2015, pp. 687–694.

[15] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya. "Fog computing: Principles, architectures, and applications". In: *Internet of things*. Elsevier, 2016, pp. 61–75.

[16] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig. "Foggy: a framework for continuous automated IoT application deployment in fog computing". In: *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. IEEE. 2017, pp. 38–45.

[17] H. R. Arkian, A. Diyanat, and A. Pourkhalili. "MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications". In: *Journal of Network and Computer Applications* 82 (2017), pp. 152–165.

[18] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara. "Cloudpath: A multi-tier cloud computing framework". In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. 2017, pp. 1–13.

[19] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. "Edge computing: Vision and challenges". In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646.

[20] M. Satyanarayanan. "The emergence of edge computing". In: *Computer* 50.1 (2017), pp. 30–39.

[21] K. Dolui and S. K. Datta. "Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing". In: *2017 Global Internet of Things Summit (GIoTS)*. IEEE. 2017, pp. 1–6.

[22] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. "The case for vm-based cloudlets in mobile computing". In: *IEEE pervasive Computing* 8.4 (2009), pp. 14–23.

[23] E. Zygmuntowicz, V. Spivak, K. Skaar, D. Collison, O. Shaldybin, M. Lucovsky, and K. Murphy. *Microcloud platform delivery system*. US Patent 8,813,065. Aug. 2014.

[24] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. "A survey of mobile cloud computing: architecture, applications, and approaches". In: *Wireless communications and mobile computing* 13.18 (2013), pp. 1587–1611.

[25] M. T. Beck, M. Werner, S. Feld, and S. Schimper. "Mobile edge computing: A taxonomy". In: *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer. 2014, pp. 48–55.

[26] A. Enayet, M. A. Razzaque, M. M. Hassan, A. Alamri, and G. Fortino. "A mobility-aware optimal resource allocation architecture for big data task execution on mobile cloud in smart cities". In: *IEEE Communications Magazine* 56.2 (2018), pp. 110–117.

[27] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani. "An open ecosystem for mobile-cloud convergence". In: *IEEE Communications Magazine* 53.3 (2015), pp. 63–70.

[28] L. Gu, D. Zeng, S. Guo, A. Barnawi, and Y. Xiang. "Cost efficient resource management in fog computing supported medical cyber-physical system". In: *IEEE Transactions on Emerging Topics in Computing* 5.1 (2015), pp. 108–119.

[29] W. Zhang, Z. Zhang, and H.-C. Chao. "Cooperative fog computing for dealing with big data in the internet of vehicles: Architecture and hierarchical resource management". In: *IEEE Communications Magazine* 55.12 (2017), pp. 60–67.

[30] S. Agarwal, S. Yadav, and A. K. Yadav. "An efficient architecture and algorithm for resource provisioning in fog computing". In: *International Journal of Information Engineering and Electronic Business* 8.1 (2016), p. 48.

[31] K. Intharawijitr, K. Iida, and H. Koga. "Analysis of fog model considering computing and communication latency in 5G cellular networks". In: *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. IEEE. 2016, pp. 1–4.

[32] V. B. C. Souza, W. Ramırez, X. Masip-Bruin, E. Marın-Tordera, G. Ren, and G. Tashakor. "Handling service allocation in combined fog-cloud scenarios". In: *2016 IEEE international conference on communications (ICC)*. IEEE. 2016, pp. 1–5.

[33] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang. "A hierarchical distributed fog computing architecture for big data analysis in smart cities". In: *Proceedings of the ASE BigData & SocialInformatics 2015*. ACM. 2015, p. 28.

[34] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand. "iFogStor: an IoT data placement strategy for fog infrastructure". In: *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*. IEEE. 2017, pp. 97–104.

[35] S. Tuli, R. Mahmud, S. Tuli, and R. Buyya. "FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing". In: *Journal of Systems and Software* (2019).

[36] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya. "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments". In: *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296.

[37] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder. "Incremental deployment and migration of geo-distributed situation awareness applications in the fog". In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM. 2016, pp. 258–269.

[38] N. Chen, Y. Yang, T. Zhang, M.-T. Zhou, X. Luo, and J. K. Zao. "Fog as a service technology". In: *IEEE Communications Magazine* 56.11 (2018), pp. 95–101.

[39] S. Tomovic, K. Yoshigoe, I. Maljevic, and I. Radusinovic. "Software-defined fog network architecture for IoT". In: *Wireless Personal Communications* 92.1 (2017), pp. 181–196.

[40] X. Sun and N. Ansari. "EdgeIoT: Mobile edge computing for the Internet of Things". In: *IEEE Communications Magazine* 54.12 (2016), pp. 22–29.

[41] S. S. Gill, R. C. Arya, G. S. Wander, and R. Buyya. "Fog-Based Smart Healthcare as a Big Data and Cloud Service for Heart Patients Using IoT". In: *International Conference on Intelligent Data Communication Technologies and Internet of Things*. Springer. 2018, pp. 1376–1383.

[42] I. Stojmenovic. "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks". In: *2014 Australasian Telecommunication Networks and Applications Conference (ATNAC)*. IEEE. 2014, pp. 117–122.

[43] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue. "Qos-aware dynamic fog service provisioning". In: *arXiv preprint arXiv:1802.00800* (2018).

[44] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen. "Vehicular fog computing: A viewpoint of vehicles as the infrastructures". In: *IEEE Transactions on Vehicular Technology* 65.6 (2016), pp. 3860–3873.

[45] M. Sookhak, F. R. Yu, Y. He, H. Talebian, N. S. Safa, N. Zhao, M. K. Khan, and N. Kumar. "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing". In: *IEEE Vehicular Technology Magazine* 12.3 (2017), pp. 55–64.

[46] G. Sun, S. Sun, H. Yu, and M. Guizani. "Towards Incentivizing Fog-Based Privacy-Preserving Mobile Crowdsensing in the Internet of Vehicles". In: *IEEE Internet of Things Journal* (2019).

[47] Z. Zhou, P. Liu, J. Feng, Y. Zhang, S. Mumtaz, and J. Rodriguez. "Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach". In: *IEEE Transactions on Vehicular Technology* 68.4 (2019), pp. 3113–3125.

[48] M. Ahmad, M. B. Amin, S. Hussain, B. H. Kang, T. Cheong, and S. Lee. "Health Fog: a novel framework for health and wellness applications". In: *The Journal of Supercomputing* 72.10 (2016), pp. 3677–3695.

[49] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, and P. Liljeberg. "Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach". In: *Future Generation Computer Systems* 78 (2018), pp. 641–658.

[50] R. Sofia and P. Mendes. "User-provided networks: consumer as provider". In: *IEEE Communications Magazine* 46.12 (2008), pp. 86–91.

[51] C. Chang, S. N. Srirama, and R. Buyya. "Indie fog: An efficient fog-computing infrastructure for the internet of things". In: *Computer* 50.9 (2017), pp. 92–98.

[52] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy. "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study". In: *IEEE Access* 5 (2017), pp. 9882–9910.

[53] F. Y. Okay and S. Ozdemir. "A fog computing based smart grid model". In: *2016 international symposium on networks, computers and communications (ISNCC)*. IEEE. 2016, pp. 1–6.

[54] C. Perera, D. S. Talagala, C. H. Liu, and J. C. Estrella. "Energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in IoT clouds". In: *IEEE Transactions on Computational Social Systems* 2.4 (2015), pp. 171–181.

[55] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, and S. Mahmoud. "SmartCityWare: A service-oriented middleware for cloud and fog enabled smart city services". In: *Ieee Access* 5 (2017), pp. 17576–17588.

[56] L. F. Bittencourt, J. Diaz-Montes, R. Buyya, O. F. Rana, and M. Parashar. "Mobility-aware application scheduling in fog computing". In: *IEEE Cloud Computing* 4.2 (2017), pp. 26–35.

[57] S. Jeong, O. Simeone, and J. Kang. "Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning". In: *IEEE Transactions on Vehicular Technology* 67.3 (2017), pp. 2049–2063.

[58] F. Zhou, Y. Wu, H. Sun, and Z. Chu. "UAV-enabled mobile edge computing: Offloading optimization and trajectory design". In: *2018 IEEE International Conference on Communications (ICC)*. IEEE. 2018, pp. 1–6.

[59] N. Mohamed, J. Al-Jaroodi, I. Jawhar, H. Noura, and S. Mahmoud. "UAVFog: A UAV-based fog computing for Internet of Things". In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE. 2017, pp. 1–8.

[60] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos. "Fog computing for sustainable smart cities: A survey". In: *ACM Computing Surveys (CSUR)* 50.3 (2017), p. 32.

[61] C. Dsouza, G.-J. Ahn, and M. Taguinod. "Policy-driven security management for fog computing: Preliminary framework and a case study". In: *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*. IEEE. 2014, pp. 16–23.

[62] J. Liu, J. Li, L. Zhang, F. Dai, Y. Zhang, X. Meng, and J. Shen. "Secure intelligent traffic light control using fog computing". In: *Future Generation Computer Systems* 78 (2018), pp. 817–824.

[63] S. Shin and G. Gu. "CloudWatcher: Network security monitoring using OpenFlow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?)" In: *2012 20th IEEE international conference on network protocols (ICNP)*. IEEE. 2012, pp. 1–6.

[64] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and

J. Turner. “OpenFlow: enabling innovation in campus networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (2008), pp. 69–74.

[65] S. Yi, Z. Qin, and Q. Li. “Security and privacy issues of fog computing: A survey”. In: *International conference on wireless algorithms, systems, and applications*. Springer. 2015, pp. 685–695.

[66] P. Hu, S. Dhelim, H. Ning, and T. Qiu. “Survey on fog computing: architecture, key technologies, applications and open issues”. In: *Journal of Network and Computer Applications* 98 (2017), pp. 27–42.

[67] M. Mukherjee, L. Shu, and D. Wang. “Survey of fog computing: Fundamental, network applications, and research challenges”. In: *IEEE Communications Surveys & Tutorials* 20.3 (2018), pp. 1826–1857.

[68] 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”. In: *IEEE Communications Surveys & Tutorials* 20.1 (2017), pp. 416–464.

[69] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian. “Resource management approaches in fog computing: a comprehensive review”. In: *Journal of Grid Computing* (2019), pp. 1–42.

[70] A. Brogi, S. Forti, C. Guerrero, and I. Lera. “How to place your apps in the fog: State of the art and open challenges”. In: *Software: Practice and Experience* 50.5 (2020), pp. 719–740.

[71] 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”. In: *IEEE Access* 6 (2018), pp. 47980–48009.

[72] S. Yi, C. Li, and Q. Li. “A survey of fog computing: concepts, applications and issues”. In: *Proceedings of the 2015 workshop on mobile big data*. ACM. 2015, pp. 37–42.

[73] A. Markus and A. Kertesz. “A survey and taxonomy of simulation environments modelling fog computing”. In: *Simulation Modelling Practice and Theory* 101 (2020), p. 102042.

[74] M. Jia, J. Cao, and W. Liang. “Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks”. In: *IEEE Transactions on Cloud Computing* 5.4 (2015), pp. 725–737.

[75] Z. Xu, W. Liang, W. Xu, M. Jia, and S. Guo. “Efficient algorithms for capacitated cloudlet placements”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.10 (2015), pp. 2866–2880.

[76] A. Ceselli, M. Premoli, and S. Secci. “Cloudlet network design optimization”. In: *2015 IFIP Networking Conference (IFIP Networking)*. IEEE. 2015, pp. 1–9.

[77] Q. Fan and N. Ansari. “Cost aware cloudlet placement for big data processing at the edge”. In: *2017 IEEE International Conference on Communications (ICC)*. IEEE. 2017, pp. 1–6.

[78] I. Martinez, A. Jarray, and A. S. Hafid. “Scalable Design and Dimensioning of Fog-Computing Infrastructure to Support Latency Sensitive IoT Applications”. In: *IEEE Internet of Things Journal* 7.6 (2020), pp. 5504–5520.

[79] M. Taneja and A. Davy. “Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm”. In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE. 2017, pp. 1222–1228.

[80] A. Karamoozian, A. Hafid, and E. M. Aboulhamid. “On the Fog-Cloud Cooperation: How Fog Computing can address latency concerns of IoT applications”. In: *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2019, pp. 166–172.

[81] M. Aazam and E.-N. Huh. “Dynamic resource provisioning through Fog micro datacenter”. In: *2015 IEEE international conference on pervasive computing and communication workshops (PerCom workshops)*. IEEE. 2015, pp. 105–110.

[82] M. Aazam, M. St-Hilaire, C.-H. Lung, and I. Lambadaris. “MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT”. In: *2016 23rd International Conference on Telecommunications (ICT)*. IEEE. 2016, pp. 1–5.

[83] M. Aazam, K. A. Harras, and S. Zeadally. “Fog computing for 5G tactile industrial Internet of Things: QoE-aware resource allocation model”. In: *IEEE Transactions on Industrial Informatics* 15.5 (2019), pp. 3085–3092.

[84] X. Peng, K. Ota, and M. Dong. “Multi-attribute based Double Auction Towards Resource Allocation in Vehicular Fog Computing”. In: *IEEE Internet of Things Journal* (2020).

[85] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, and F. Desprez. “Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM. 2018, pp. 751–760.

[86] L. P. Kaelbling, M. L. Littman, and A. W. Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[87] Y. Wei, F. R. Yu, M. Song, and Z. Han. “Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning”. In: *IEEE Internet of Things Journal* 6.2 (2018), pp. 2061–2073.

[88] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo. “Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications”. In: *IEEE Transactions on Industrial Informatics* 15.2 (2018), pp. 976–986.

[89] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu. “Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system”. In: *IEEE Transactions on Computers* 65.12 (2016), pp. 3702–3712.

[90] H. Zhang, Y. Xiao, S. Bu, D. Niyato, R. Yu, and Z. Han. "Fog computing in multi-tier data center networks: A hierarchical game approach". In: *2016 IEEE international conference on communications (ICC)*. IEEE. 2016, pp. 1–6.

[91] Q. Wang. "PFogSim: A Simulator for Evaluating Dynamic and Layered Fog Computing Environments". In: (2019).

[92] M. Ali, N. Riaz, M. I. Ashraf, S. Qaisar, and M. Naeem. "Joint cloudlet selection and latency minimization in fog networks". In: *IEEE Transactions on Industrial Informatics* 14.9 (2018), pp. 4055–4063.

[93] V. Cardellini, V. Grassi, F. L. Presti, and M. Nardelli. "On QoS-aware scheduling of data stream applications over fog computing infrastructures". In: *2015 IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2015, pp. 271–276.

[94] Y. Sun, M. Peng, and S. Mao. "Deep reinforcement learning-based mode selection and resource management for green fog radio access networks". In: *IEEE Internet of Things Journal* 6.2 (2018), pp. 1960–1971.

[95] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In: *Software: Practice and experience* 41.1 (2011), pp. 23–50.

[96] L. Ni, J. Zhang, C. Jiang, C. Yan, and K. Yu. "Resource allocation strategy in fog computing based on priced timed petri nets". In: *ieee internet of things journal* 4.5 (2017), pp. 1216–1228.

[97] F. A. Salaht, F. Desprez, A. Lebre, C. Prud'Homme, and M. Abderrahim. "Service Placement in Fog Computing Using Constraint Programming". In: *IEEE INTERNATIONAL CONFERENCE ON SERVICES COMPUTING*. 2019.

[98] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos. "Fog Based Framework for IoT Service Provisioning". In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2019, pp. 1–6.

[99] Y. Jiang, Z. Huang, and D. H. Tsang. "Challenges and solutions in fog computing orchestration". In: *IEEE Network* 32.3 (2018), pp. 122–129.

[100] M. S. de Brito, S. Hoque, T. Magedanz, R. Steinke, A. Willner, D. Nehls, O. Keils, and F. Schreiner. "A service orchestration architecture for fog-enabled infrastructures". In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2017, pp. 127–132.

[101] N. Chen and S. Clarke. "A dynamic service composition model for adaptive systems in mobile computing environments". In: *International Conference on Service-Oriented Computing*. Springer. 2014, pp. 93–107.

[102] N. Madan, A. W. Malik, A. U. Rahman, and S. D. Ravana. "On-demand resource provisioning for vehicular networks using flying fog". In: *Vehicular Communications* (2020), p. 100252.

[103] K. Hong, D. Lillethun, U. Ramachandran, B. Ottenwälder, and B. Koldehofe. "Mobile fog: A programming model for large-scale applications on the internet of things". In: *Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing*. 2013, pp. 15–20.

[104] S. Yangui, P. Ravindran, O. Bibani, R. H. Glitho, N. B. Hadj-Alouane, M. J. Morrow, and P. A. Polakos. "A platform as-a-service for hybrid cloud/fog environments". In: *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*. IEEE. 2016, pp. 1–7.

[105] D. Soni and A. Makwana. "A survey on mqtt: a protocol of internet of things (iot)". In: *International Conference On Telecommunication, Power Analysis And Computing Techniques (ICTPACT-2017)*. 2017.

[106] A. Lara, A. Kolasani, and B. Ramamurthy. "Simplifying network management using software defined networking and OpenFlow". In: *2012 IEEE International Conference on Advanced Networks and Telecommunciations Systems (ANTS)*. IEEE. 2012, pp. 24–29.

[107] J. Hasenburg, S. Werner, and D. Bermbach. "Supporting the evaluation of fog-based IoT applications during the design phase". In: *Proceedings of the 5th Workshop on Middleware and Applications for the Internet of Things*. 2018, pp. 1–6.

[108] J. Hasenburg, S. Werner, and D. Bermbach. "FogExplorer". In: *Proceedings of the 19th International Middleware Conference (Posters)*. 2018, pp. 1–2.

[109] J. Hasenburg. *FogExplorer Inrastructure as Code*. https : / / github . com / OpenFogStack / FogExplorer. Accessed: 2020-07-25.

[110] M. I. Naas, J. Boukhobza, P. R. Parvedy, and L. Lemarchand. "An extension to ifogsim to enable the design of data placement strategies". In: *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*. IEEE. 2018, pp. 1–8.

[111] A. Khanna and R. Tomar. "IoT based interactive shopping ecosystem". In: *2016 2nd International Conference on Next Generation Computing Technologies (NGCT)*. IEEE. 2016, pp. 40–45.

[112] M. Seufert, B. K. Kwam, F. Wamser, and P. Tran-Gia. "Edgenetworkcloudsim: Placement of service chains in edge clouds using NetworkCloudSim". In: *2017 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2017, pp. 1–6.

[113] B. K. Kwam. *An extension of NetworkCloudSim to simulate the Edge Cloud*. https://github.com/lsinfo3/EdgeNetworkCloudSim. Accessed: 2020-05-02.

[114] M. I. Naas. *iFogSimWithDataPlacement on GitHub*. https : / / github . com / medislam / iFogSimWithDataPlacement. Accessed: 2020-05-02.

[115] X. Liu, L. Fan, J. Xu, X. Li, L. Gong, J. Grundy, and Y. Yang. "FogWorkflowSim: An Automated Simulation Toolkit for Workflow Performance Evaluation

in Fog Computing". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 1114–1117.

[116] W. Chen and E. Deelman. "Workflowsim: A toolkit for simulating scientific workflows in distributed environments". In: *2012 IEEE 8th International Conference on E-Science*. IEEE. 2012, pp. 1–8.

[117] M. M. Lopes, W. A. Higashino, M. A. Capretz, and L. F. Bittencourt. "Myifogsim: A simulator for virtual machine migration in fog computing". In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*. 2017, pp. 47–52.

[118] M. M. Lopes. *MyiFogSim on GitHub*. https://github.com/marciocomp/myifogsim. Accessed: 2020-04-30.

[119] C. Puliafito, D. M. Gonçalves, M. M. Lopes, L. L. Martins, E. Madeira, E. Mingozzi, O. Rana, and L. F. Bittencourt. "MobFogSim: Simulation of mobility and migration for fog computing". In: *Simulation Modelling Practice and Theory* 101 (2020), p. 102062.

[120] T. Qayyum, A. W. Malik, M. A. K. Khattak, O. Khalid, and S. U. Khan. "FogNetSim++: A toolkit for modeling and simulation of distributed fog environment". In: *IEEE Access* 6 (2018), pp. 63570–63583.

[121] T. Qayyum. *A Toolkit to simulate distributed fog computing environment*. https://github.com/rtqayyum/fognetsimpp. Accessed: 2020-04-30.

[122] A. Varga. "OMNeT++". In: *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[123] I. Lera, C. Guerrero, and C. Juiz. "YAFS: A simulator for IoT scenarios in fog computing". In: *IEEE Access* 7 (2019), pp. 91745–91758.

[124] C. Mechalikh, H. Taktak, and F. Moussa. "PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments". In: *The 2019 International Conference on High Performance Computing Simulation*. IEEE, 2019, pp. 700–707.

[125] C. Mechalikh. *PureEdgeSim: A Simulation Toolkit for Performance Evaluation of Cloud, Fog, and Pure Edge Computing Environments*. https : / / github . com/CharafeddineMechalikh/PureEdgeSim. Accessed: 2020-04-30.

[126] S. Forti, A. Pagiaro, and A. Brogi. "Simulating FogDirector Application Management". In: *Simulation Modelling Practice and Theory* 101 (2020), p. 102021.

[127] S. Forti. *A Simulator of CISCO FogDirector Application Management*. https://github.com/di-unipi-socc/FogDirSim. Accessed: 2020-05-02.

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

[129] R. Mayer, L. Graser, H. Gupta, E. Saurez, and U. Ramachandran. "Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures". In: *2017 IEEE Fog World Congress (FWC)*. IEEE. 2017, pp. 1–6.

[130] I. Behnke, L. Thamsen, and O. Kao. "Héctor: A Framework for Testing IoT Applications Across Heterogeneous Edge and Cloud Testbeds". In: *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion*. 2019, pp. 15–20.

[131] J. Hasenburg, M. Grambow, E. Grünewald, S. Huk, and D. Bermbach. "MockFog: Emulating fog computing infrastructure in the cloud". In: *2019 IEEE International Conference on Fog Computing (ICFC)*. IEEE. 2019, pp. 144–152.

[132] J. Hasenburg. *MockFog Inrastructure as Code*. https://github.com/OpenFogStack/MockFog-IaC. Accessed: 2020-07-24.

[133] L. Graser. *EmuFog on GitHub*. https://github.com/emufog/emufog. Accessed: 2020-07-24.

[134] A. Forster and A. L. Murphy. "FROMS: Feedback routing for optimizing multiple sinks in WSN with reinforcement learning". In: *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*. IEEE. 2007, pp. 371–376.

[135] R. Arroyo-Valles, R. Alaiz-Rodriguez, A. Guerrero-Curieses, and J. Cid-Sueiro. "Q-probabilistic routing in wireless sensor networks". In: *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*. IEEE. 2007, pp. 1–6.

[136] T. Hu and Y. Fei. "QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks". In: *IEEE Transactions on Mobile Computing* 9.6 (2010), pp. 796–809.

[137] A. A. Bhorkar, M. Naghshvar, T. Javidi, and B. D. Rao. "Adaptive opportunistic routing for wireless ad hoc networks". In: *IEEE/ACM Transactions On Networking* 20.1 (2011), pp. 243–256.

[138] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner. "Optimized IoT service placement in the fog". In: *Service Oriented Computing and Applications* 11.4 (2017), pp. 427–443.

[139] A. Medina, A. Lakhina, I. Matta, and J. Byers. "BRITE: An approach to universal topology generation". In: *MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE. 2001, pp. 346–353.

[140] H. Gupta. *iFogSim on GitHub*. https://github.com/Cloudslab/iFogSim. Accessed: 2020-05-02.

[141] A. A. A. Ateya, A. Muthanna, R. Kirichek, M. Hammoudeh, and A. Koucheryavy. "Energy-and latency-aware hybrid offloading algorithm for UAVs". In: *IEEE Access* 7 (2019), pp. 37587–37600.