Context-Aware Routing in Fog Computing Systems

Vasileios Karagiannis, Pantelis A. Frangoudis, Schahram Dustdar, and Stefan Schulte

Abstract—Fog computing enables the execution of IoT applications on compute nodes which reside both in the cloud and at the edge of the network. To achieve this, most fog computing systems route the IoT data on a path which starts at the data source, and goes through various edge and cloud nodes. Each node on this path may accept the data if there are available resources to process this data locally. Otherwise, the data is forwarded to the next node on path. Notably, when the data is forwarded (rather than accepted), the communication latency increases by the delay to reach the next node. To avoid this, we propose a routing mechanism which maintains a history of all nodes that have accepted data of each context in the past. By processing this history, our mechanism sends the data directly to the closest node that tends to accept data of the same context. This lowers the forwarding by nodes on path, and can reduce the communication latency. We evaluate this approach using both prototype- and simulation-based experiments which show reduced communication latency (by up to 23%) and lower number of hops traveled (by up to 73%), compared to a state-of-the-art method.

Index Terms—Fog computing, Edge computing, Internet of Things, Context-aware, IoT applications

1 Introduction

The proliferation of the Internet of Things (IoT) has increased significantly the number of Internet-connected devices, and consequently the amount of data that travels through the Internet [1], [2]. This data can overwhelm the network in the centralized cloud paradigm, potentially causing latency- and bandwidth-related issues [3]. To cope with such issues, fog computing proposes the utilization of both cloud and edge compute nodes [4], [5]. This may hinder the accumulation of data in one central location, reduce the communication latency, and improve bandwidth utilization because the computations of the IoT data can also be performed close to the data sources [6].

To achieve such benefits, many fog computing systems route the IoT data on a path which starts at the data source, and goes through various edge and cloud compute nodes until the data is accepted for processing [7], [8], [9]. The compute nodes on this path are usually assumed to be ordered based on their proximity to the data source. Thus, the first compute nodes are able to process the IoT data with low communication latency, but this latency increases when nodes farther along the path are utilized for the processing [10]. This way, the compute node closest to the data source with adequate available computational resources, is utilized for processing the IoT data with low latency. Typically, the compute nodes at the edge of the network integrate limited computational resources [11], [12]. As a result, only a fraction of the IoT data is accepted by nearby compute nodes (i.e., the first nodes on path), while the rest of the data is more likely to be processed in a remote cloud node [13].

- *V. Karagiannis, P.A. Frangoudis, and S. Dustdar are with the Distributed Systems Group of TU Wien, Vienna, Austria. S. Schulte is with the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things at TU Wien.*
  E-mail: {v.karagiannis, p.frangoudis, dustdar, s.schulte} @dsg.tuwien.ac.at

Manuscript received Month Date, Year; revised Month Date, Year.

In such systems, every time a compute node forwards the IoT data, the communication latency is increased by both the time required by a compute node to decide whether to accept the data or not, and the time needed to reach the next node. Therefore, this latency accumulates while the IoT data is forwarded by nodes on path. Furthermore, routing the data through intermediate nodes (rather than directly) to the compute node that accepts it, may increase the number of network hops traveled, and potentially the utilization of network bandwidth [14].

To counter these problems, we propose a mechanism for context-aware routing in fog computing systems. Instead of routing the IoT data along the path of compute nodes until a node with adequate resources is found, our mechanism aims at sending the data directly to that node. This eliminates the overhead of sending the data through intermediate nodes, and consequently lowers the communication latency of performing IoT computations. Since the target node is not known a priori, our mechanism relies on the context of the data, and on information from previous transmissions. Based on these, the proposed mechanism first examines which compute node has accepted data of the same context in the past. Then, the IoT data is routed on a path that starts from that node, thereby bypassing other compute nodes that typically forward such data due to not having enough available computational resources to process it.

Our contributions include a system model for executing IoT applications in fog computing systems, and a mechanism for realizing the proposed context-aware routing. In addition, we implement and evaluate concrete strategies that can be used in our mechanism for selecting nodes with available resources, in particular, based on predictive methods such as Reinforcement Learning (RL). To show the benefits of our approach, we build a fog computing system using real-world geographically distributed compute nodes, and we perform an extensive prototype- and simulation-based evaluation considering a smart energy application.

According to our results, context-aware routing lowers the communication latency of sending IoT data by up to 23%, and reduces the hop count by up to 73%, compared to a state-of-the-art method.

The rest of this paper is organized as follows: In Section 2 we discuss related work. Afterwards, in Section 3 we present a system model for executing IoT applications in fog computing systems, and in Section 4 we propose a novel mechanism for context-aware routing in such systems. Subsequently, Section 5 presents an evaluation of the proposed mechanism by comparing it to a baseline method, and Section 6 concludes this paper and proposes promising research directions for future work.

2 Related Work

There are various approaches in the literature that propose mechanisms for enabling compute nodes of fog computing systems to either accept IoT data and process it, or to forward the data to other compute nodes. For instance, Tong et al. [8] propose a mechanism for forwarding peak workloads from mobile devices to compute nodes towards the cloud in order to increase the computation capacity of a system which includes various edge and cloud nodes. Chekired et al. [15] present a mechanism for processing industrial IoT data based on priorities such that high priority data is processed on compute nodes close to the edge, and low priority data is forwarded towards the cloud. Ascigil et al. [7] propose a system with various compute nodes on the path from the edge to the cloud, and design a mechanism based on deadlines, to either accept or forward a workload. Mortazavi et al. [9] design a platform which assumes that network devices between the edge and the cloud act as compute nodes, and are able to process the data on a path towards the cloud.

Notably, the aforementioned approaches assume that the data is routed on a path from the edge towards the cloud, and propose mechanisms to enable the compute nodes on path to either accept or forward this data. However, none of these approaches considers bypassing compute nodes on path to avoid the extra overhead. For this reason, the work at hand presents a mechanism that takes into account the context of the IoT data for avoiding busy compute nodes on path in order to process the data with reduced communication latency, as discussed in Section 1. Other approaches from the literature which leverage the context of the IoT data in order to perform IoT computations efficiently, are discussed below.

Mahmud et al. [16] propose an application placement policy for fog computing systems which aim at facilitating industrial applications. This policy considers the context of the IoT data for coordinating the workload of the IoT devices with the compute capacities of the nodes in order to minimize the service delivery time. However, the authors do not take into account that the context of the IoT data can be further leveraged for routing the data to compute nodes with available computational resources, which is the scope of our work.

Mononen et al. [17] discuss a system for executing applications on distributed compute nodes. In this system, the compute nodes utilize a mechanism that leverages the context of the data to avoid sending unnecessary information to the cloud in order to reduce the network traffic. In our work, we also allow sending data only to nearby compute nodes. However, in addition to that, we enable the IoT data to bypass nearby nodes, in case these nodes are not able to execute the required computations. This may lower the overhead of forwarding IoT data through the various compute nodes of a fog computing system, as discussed in Section 1.

Roy et al. [18] present an approach for storing only the context of the IoT data in nearby compute nodes, i.e., not the actual values of the data. This context is then shared among all the nearby compute nodes periodically, in order to facilitate unified IoT applications. However, the authors do not consider using the context for sending the IoT data directly to compute nodes with available computational resources, which is the scope of our work.

Akbar et al. [19] present an architecture for stream processing in the IoT. The authors propose a mechanism that utilizes the context of the IoT data, which is acquired from previous transmissions, in order to define threshold values needed by complex event processing engines. In our work, we also use previous transmissions for defining the context of the data. However, in contrast to the work by Akbar et al., we use the context to change the routing paths of the data, and to improve the efficiency of the system.

Wiener et al. [20] describe a conceptual architecture for context-aware stream processing in fog computing systems. In this work, the authors propose relocating the applications according to the changes in the context of the IoT data. In our work, instead of relocating the applications, we use the context of the data to change the routing paths, and to send the data directly to compute nodes that can perform the required computations.

To summarize the discussion, there are various approaches in the literature which leverage the context of the data in order to improve performing computations in the IoT. However, to the best of our knowledge, none of these approaches considers a context-aware routing mechanism for fog computing systems. In the work at hand, we design and implement a context-aware routing mechanism that provides reduced communication latency and improved bandwidth utilization compared to a state-of-the-art method which is based on previously discussed approaches (such as [9] and [7]).

Finally, there is related work from the field of cloud computing, which considers a centralized scheduler that maintains a global view of the system, i.e., the addresses of all the candidate compute nodes are known a priori [21]. Such a scheduler is able to perform application placement, usually based on optimization logic, so that each application is instantiated on a specific compute node, and the data is sent to that node directly (i.e., without traveling on a path of multiple nodes). Notably, such approaches have also been applied to computing systems that include both cloud and edge compute nodes [22]. In such systems however, maintaining a global view may become infeasible or prohibitively costly, e.g., due to their scale (which may be too large to be centrally coordinated) [23], or their dynamicity (which results in nodes joining/leaving concurrently) [24],

[25], [26]. To avoid such issues, many approaches consider that fog computing systems are not centrally orchestrated, and no node maintains a global view of the system [8], [9]. Instead, each node maintains only few neighbors, i.e., a limited view of the system, and the data is propagated in the system on a path through these neighbors [27]. For such systems, we design a mechanism that enables the IoT data to be sent directly to appropriate compute nodes even without a central scheduler component. To achieve that, we leverage the context of the IoT data, and a history of previous transmissions that stores which context is typically accepted by each node.

3 System Model

In this section, we present a system model for executing IoT applications in nearby and remote compute nodes based on fog computing principles. This system model includes the arrangement of the compute nodes, the flow of the data with regard to the execution of the applications, and the type of IoT applications that we consider. Specifically, our model is based on the fog computing model provided by the OpenFog Reference Architecture [28], which has been adopted in the IEEE 1934-2018 standard. We base our system model on this architecture due to the wide variety of applicable use cases which may target smart cities and smart grids, intelligent hospitals and healthcare, intelligent factories and logistics optimization, among others [28], [29].

In our system model, there can be various compute nodes which span from the edge of the network where the IoT devices reside, until the cloud. We refer to the entirety of the participating compute nodes and devices as a fog computing system [27]. An example of a fog computing system is depicted in Fig. 1.

As shown in this figure, a fog computing system can include multiple compute nodes which reside either in the cloud or at the edge of the network [30]. The cloud compute nodes are nodes in data centers that may be located far away from the IoT devices. The edge compute nodes represent available computational resources at the network edge, e.g., access points, base stations, or specialized edge nodes offered by cloud providers (such as the edge zones by Microsoft [31]). Commonly, all of these compute nodes communicate with each other by forming hierarchies over the Internet [27], [30], as shown in Fig. 1.

The IoT devices are located at the bottom of the hierarchy, and physically close to the edge compute nodes [32]. These devices integrate sensors and/or actuators in order to sense and/or interact with the surrounding environment [5]. The IoT devices are usually resource-constrained, and may not integrate enough computational resources to implement the necessary communication protocols for interacting with the compute nodes directly (e.g., using an application layer protocol such as HTTP) [33]. For this reason, a gateway is used [28]. This gateway provides two interfaces: One interface is for the communication with the IoT devices (commonly using low-power wireless protocols such as Zigbee or Bluetooth [34]). The other interface is used for the communication with compute nodes over the Internet [35]. The gateway may also be able to act as a compute node, in which case the second interface might not be used if only the local computational resources are utilized for executing the required computations.

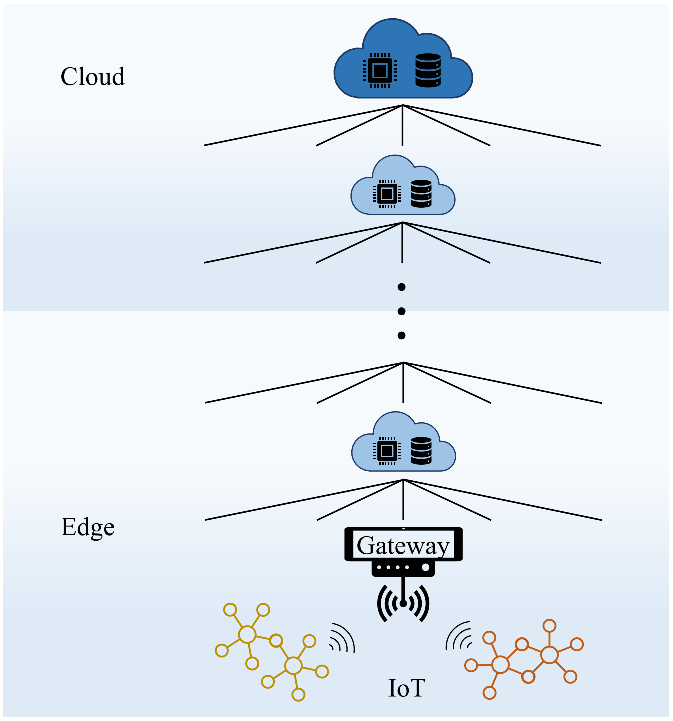


Fig. 1: A fog computing system with edge and cloud compute nodes.

A fog computing system may consist of multiple compute nodes, gateways, and IoT devices. Notably, Fig. 1 depicts the communication between IoT devices and one gateway, and also the gateway and compute nodes. This is done for simplicity because other devices and nodes form similar hierarchical paths towards the cloud. Each layer of the hierarchy provides additional computational resources [28]. Typically, the compute nodes close to the IoT devices provide limited computational resources with low communication latency [28]. The compute nodes which are farther along the path, on the other hand, are able to provide more or even (virtually) unlimited computational resources. Nevertheless, the communication latency to reach these compute nodes also increases due to the longer network paths [36]. Because of the virtually unlimited resources of remote cloud nodes, we assume that the last compute node on path is always a cloud compute node which is able to accept any application request [14].

3.1 IoT Data Flow

The flow of the data in this system model starts at the IoT devices which generate the IoT data using sensors [37], [38]. The IoT data is then sent to the gateway which encapsulates this data into an application request. An application request consists of the necessary information to request the execution of a computation from a compute node [39]. This includes an application identifier indicating the specific application to be instantiated, and the IoT data to be used as input for this application. The application identifier is selected based on the identifier of the sensor that generated the data. The association between sensors and applications is configured in the gateway when deploying the IoT devices. This configuration can be altered when new IoT devices and/or new applications are deployed. Thus, the gateway receives the IoT data, creates an application request by adding an application identifier, and sends this request to a compute node in proximity.

Each compute node in a fog computing system is able to receive application requests through an interface. Upon the arrival of such a request, the node examines the utilization of its local computational resources. Based on this, the node decides whether to instantiate an application (e.g., using software containers or virtual machines) and process the data, or to forward the application request to the next compute node upwards the hierarchy (in a depth-first manner [8]). Thus, similar to related work (e.g., [7], [8], [27]), we assume that each compute node (and also the gateway) is able to communicate with a node in the higher hierarchical layer.

When a compute node accepts an application request and the required files to instantiate the application are not available locally, the node downloads these files (which are located using the application identifier) from a repository which contains all the applications (e.g., Docker hub). We assume that every node needs to download the files only once at the beginning of each application [40]. For this reason, we consider that the download overhead is negligible.

If a node cannot download the requested application (e.g., due to limited permissions), the application request is forwarded to the next node. For example, applications related to monitoring and storing, may be eligible for execution only in the cloud, because the cloud can act as a point for central monitoring [41]. The application which handles the final processing of the data, sends the final output to the gateway (for actuating commands) and/or to the cloud (for monitoring and storing purposes).

The output of executing an application request can be: raw information to be stored in the cloud (e.g., for monitoring), an actuation command to be sent to an IoT device, or a new application request. This depends on the logic and function of each application [27]. In case the output is an application request, this request is first examined locally, and if there are not enough available computational resources, it is forwarded to the next node upwards the hierarchy. The reason that the request is forwarded upwards (i.e., towards the cloud) is that since a compute node has received an application request, it is expected that the compute nodes lower in the hierarchy are less likely to accept the new request (e.g., due to being busy) than the compute nodes upwards.

3.2 Application Model

The core of the previously discussed system model (i.e., with compute nodes that form paths of edge and cloud compute nodes) has been deemed suitable for addressing various use cases such as: IoT analytics, wearable cognitive assistance, augmented reality, image processing, intelligent transportation systems, interactive networked gaming, and industrial manufacturing [7], [8], [9], [15]. To represent such use cases, we consider an application model that consists of multiple applications which can be executed in a distributed manner, and work towards a common goal [36], as shown in Fig. 2. Initially, a sensor, e.g., a gas meter in case of an IoT smart energy application, generates measurements. These measurements are then processed by various applications which may be related to anomaly detection, cost optimization, etc. The output of these applications can be a command such as: to store information in the cloud, to turn on/off the heating, to send a smartphone notification (e.g., regarding leaks), etc. In our system model, the measurements are sent to the gateway which sends this data on a path towards the cloud. When an actuation command is generated (by an application), it is sent back to the gateway and subsequently, to the target IoT device. In this manner, a wide range of use cases that involve input from sensors, processing from different applications, and output related to commands (e.g., send notification, store, or actuate), can be represented by our application model.

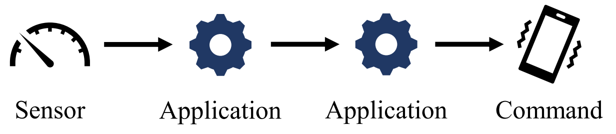


Fig. 2: Example of the application model.

Notably, in this system model the application provider is responsible for providing, maintaining, and configuring the IoT devices, the gateway, and the compute nodes. The same model can also be applicable to a system which uses only a remote cloud compute node, i.e., by following the centralized cloud computing paradigm. Nevertheless, there are various aspects of this model which favor a system that utilizes distributed nearby and remote compute nodes. For example: the way that application execution is requested (i.e., using the input data along with an application identifier), the use of multiple applications which can be executed on different compute nodes, the permissions of the compute nodes to download and execute different applications, the use of a repository which hosts all the files of the applications, and is accessible from the compute nodes. Thus, when using such a system model, an application provider is expected to be owning/leasing/renting and employing various distributed compute nodes.

4 A Routing Mechanism For Fog Computing

Based on the system model discussed in Section 3, in the following we present a novel mechanism which considers the context of the IoT data in order to route application requests to the compute nodes of a fog computing system. To this end, first we discuss the conceptual foundation of the proposed mechanism in Section 4.1, and after that in Section 4.2, we describe how the proposed mechanism can be implemented. Finally, Section 4.3 discusses a concrete predictive method that can be used within our mechanism in order to select the closest compute with available resources.

4.1 Conceptual Foundation

Unlike alternative approaches from the literature (discussed in Section 2), which route the IoT data on a path from the edge towards the cloud as shown in Fig. 3a, our approach aims at sending the IoT data directly to the closest compute node with available computational resources. To achieve that, we design a routing mechanism for the gateway. According to this mechanism, the gateway determines which compute node is more likely to accept an application request. Then, based on this information, the gateway sends the application request to that compute node directly. Thus, as shown in Fig. 3b, the gateway of the proposed approach is able to send data directly to each compute node of the system. In case the selected node cannot accept the application request (e.g., due to dynamic factors such as the availability of computational resources), the proposed mechanism falls back to the traditional routing approach (as shown in Fig. 3a). This means that after the initial transmission of an application request, i.e., from the gateway to a selected compute node, this compute node either accepts the request, or forwards to the next node upwards the hierarchy, as discussed in Section 3.

The aim of this logic aligns with two prime goals of fog computing [6], [25]: *i*) To reduce the latency of sending IoT data. *ii*) To improve the utilization of the available network bandwidth. Considering these two goals, we make the following key observations regarding traditional routing (i.e., when routing the data on a path towards the cloud):

- When the IoT data is sent towards the cloud, it is expected that a compute node close to the gateway accepts and processes the data. However, considering that the compute nodes at the edge of the network have limited computational resources [42], and that the amount of data and computations from the IoT keeps increasing significantly [1], it is likely that only a fraction of the IoT data is actually processed at the edge of the network. The rest of the data, trying to find the closest compute node with available computational resources, is forwarded closer to the cloud. Thus, in traditional routing, a big part of the IoT data is processed by compute nodes with communication latency that has been increased by all the previous nodes that examined and forwarded the application requests due to not having available computational resources.
- When an application request is sent, e.g., from a gateway A to a compute node D, this request is sent based on a routing algorithm, e.g., using the border gateway protocol that finds the best routing path from A to D [43]. However, when a request is routed through other nodes, e.g., from A to B to C to D, even though each transmission (e.g., A to B, B to C, and C to D) is routed through the best path, the transmission to the final node (i.e., from A to D) may include detours in case B and C do not exist within the best path from A to D. Thus, if the utilized network path to a compute node that accepts an application request includes detours, it is likely that both the communication latency and the bandwidth utilization are increased because the best path has not been followed.

Our approach aims at enabling the gateway to send each application request directly to the closest compute node with available computational resources. This may help: *i*) to limit the communication latency due to avoiding the forwarding by nodes on path, and *ii*) to improve bandwidth utilization by avoiding network detours.

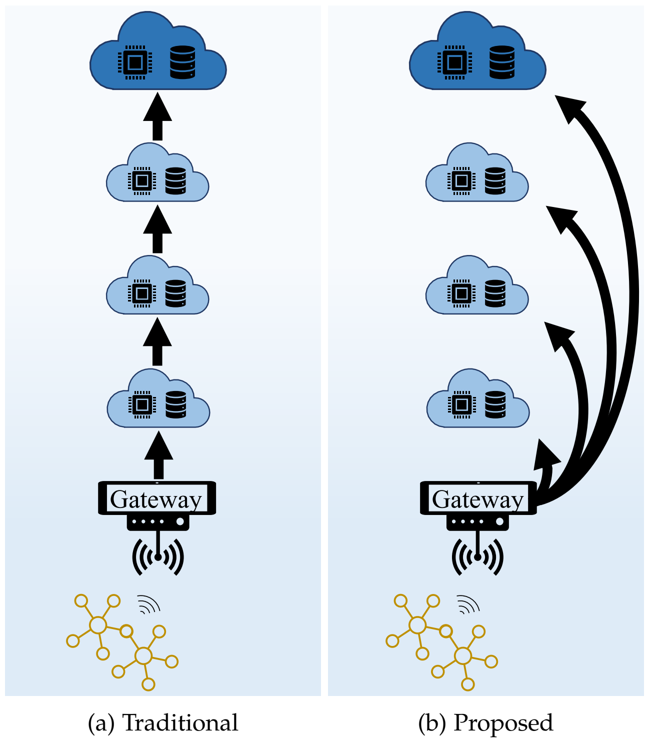


Fig. 3: Routing approaches in fog computing systems: (a) shows a gateway which sends data through compute nodes, whereas (b) shows a gateway which is able to send data directly to each compute node.

4.2 Context-Aware Routing

4.2.1 Prerequisites

To design a routing mechanism for fog computing according to the logic presented in Section 4.1, we leverage the context of the IoT data. To define context, we build upon the literature of context-aware computing, in which the applications are able to use information gleaned from different parts of the system in order to adapt their behavior [44]. Thus, by leveraging such information for routing, which is the goal of our work, we aim at adapting the routing paths of the data, in order to improve the network performance of fog computing systems [44].

In context-aware computing systems, it is usually the sensors that provide information which is considered as context [45]. Despite that, it has been observed that the notion of context in a system, may be consisting of different information that is correlated and interdependent. Therefore, in some cases (such as for learning and predicting purposes, or for decision making) the context should consider information from different parts of the system [46]. In our system, the context should represent the amount of computational resources which are needed for accepting an application request. This way, there can be a relation between the context of an application request, and the compute node that typically accepts it, i.e., the node that usually has enough available resources to accept it.

For this reason, in this work we define as context the combination of the sensor identifier (of the sensor that generated the data), and the application identifier (of the application that is needed for processing the data). These are both common parameters in IoT systems [47]. We define the context in this manner because the sensor data alone may not hold enough information to indicate the required resources, since this also depends on the tasks of the application (e.g., for the same input data, an application that applies a filter may need less resources than an application that trains an artificial neural network). Similarly, an application identifier alone may also not be enough to indicate the required resources, because this also depends on the input data. The amount of input data used in each application request, is assumed to be consistent for each application, e.g., an application that applies a filter on an image, is expected to always receive the same number of images per application request. We make this assumption so that the amount of data per application request, does not affect (significantly) the required resources to accept this request.

This way, by combining the sensor and application identifiers, we consider that the context can be representative of the amount of computational resources required for executing an application request. According to this context definition, we design a mechanism for the gateway, which keeps a history of the compute nodes that accept application requests, along with the context that each compute node accepts. This way, every time the gateway is about to send a new application request, the history is examined in order to find the compute node that usually accepts requests of the same context, i.e., the compute node that usually has sufficient available computational resources for executing an application request of the same context.

Presumably, our work relies on the assumption that a compute node which accepts data of a particular context, is likely to accept such data again in the future. We claim that this a reasonable assumption in fog computing systems because the farther along the path the IoT data travels, compute nodes with additional computational resources are found [28]. Therefore, the probability that an application request is accepted, increases while the data is forwarded along the path. Notably, this probability depends also on the current load of each compute node, and the amount of computational resources needed for accepting an application request. Since the load of the compute nodes is not known at the gateway, we rely on the latter. Thus, we leverage the context of the IoT data, because it is considered representative of the computational resources needed for accepting an application request. This is discussed further in Section 5.2.2 with results which support that in a path with an increasing probability of accepting the data, it is the same compute nodes that tend to accept application requests of the same context.

4.2.2 Proposed Approach

Fig. 4 shows the high-level architecture of the components that we place in the gateway in order to implement the proposed mechanism. The *application identifiers* component stores the identifiers of the applications that are needed for processing the data from each sensor. This component is assumed to be preconfigured, as discussed in Section 3. The *compute nodes* component stores the addresses of compute nodes along with the context of the application requests that each compute node has accepted in the past. Initially, this component is assumed to have the address of a compute node in proximity, as discussed in Section 3.

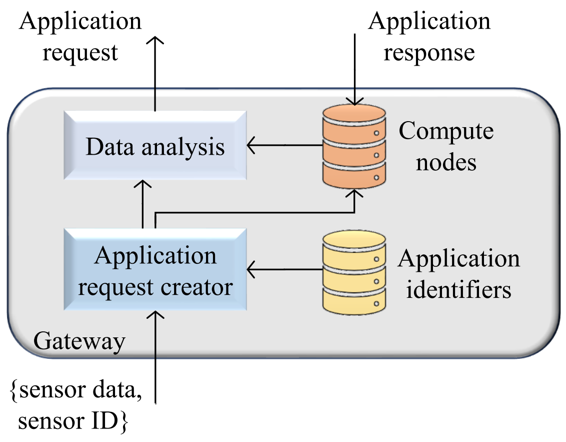


Fig. 4: The high-level architecture of the proposed mechanism.

The functionality of the proposed mechanism is triggered by sending a set of data that requires processing, along with the sensor identifier, to the *application request creator*, as shown in Fig. 4. It is also possible that the set of data consists of subsets of data that have been generated by more than one sensor. In this case, each subset is accompanied by a sensor identifier. The application request creator compares the sensor identifier(s) of the received data with the sensor identifiers of the application identifiers component, and creates an application request with the match (by adding the application identifier, as discussed in Section 3). The application request is then sent to the *data analysis* component. At the same time, the context of this request (i.e., sensor and application identifiers) is sent to the compute nodes component. If an application request of the same context has not been transmitted before, the data analysis component pulls the address of a node in proximity from the compute nodes component, and sends the application request to that node.

Then, the application request travels through the compute nodes of the system until a node with available computational resources accepts it. Upon acceptance, the compute node instantiates an application as discussed in Section 3, and responds with its address (i.e., the address of the node that accepted the request) which we refer to as the application response. Subsequently, the application response is sent back to the gateway, as shown in Fig. 4. The gateway stores the application response in the compute nodes component, as the address of the compute node that accepted the context that was stored in the compute nodes component when the application request was created. Thus, the gateway stores the context of each application request before sending it, and adds the address of the compute node that accepts it, as soon as this node sends back the application response.

In case an application request of a specific context has been transmitted before, the data analysis component sends the request to one of the compute nodes that have accepted requests of the same context in the past. The decision on which one of these nodes should be selected, can be made based on various strategies such as: most recently used, most frequently used, or using predictive methods. Notably, current machine learning approaches from the literature which aim at selecting appropriate compute nodes for deploying applications, may also make suitable strategies [48]. In particular, we consider RL to be an intuitive choice for the logic of the data analysis component, because the application response contains the node that accepted each application request. This information can then be used for determining the reward/penalty in a RL-based model [49]. We elaborate further on this strategy in Section 4.3, which provides the concrete logic of an RL-based data analysis component for our mechanism.

Interestingly, our mechanism stores the minimum amount for information needed to be able to send application requests to compute nodes based on context. This information includes only the address of candidate compute nodes, and the context of the application requests. While alternative approaches (discussed in Section 2) may also use additional properties such as the location of the nodes, or their resource capacities [8], [15], we store less information in order to make our mechanism more practical for gateways which do not have ample memory resources. However, such properties are in fact considered implicitly in our mechanism. For example, the locations of nodes are taken into account by using latency as a proximity measure, and by preferring nodes which respond with less latency (i.e., nearby nodes). Also, the resource capacities of the nodes are considered because only the nodes with sufficient resources are stored in the gateway.

In our approach, the data analysis component may also cross the information of different contexts. For instance, when there are many candidate nodes which grade equally for a specific context, the ones that have been used recently for other contexts may be excluded to avoid potentially busy nodes. In case the selected node is unresponsive (e.g., due to temporary/permanent disconnection or failure), another node can be selected based on the same strategy as long as one exists. Otherwise, the proposed mechanism falls back to the traditional approach, and sends the application request to the compute node in proximity.

Since fog computing systems can be volatile, and compute nodes may be added/removed temporarily or permanently at any time [25], we use a system parameter to reset the compute nodes component to its initial state periodically (e.g., based on time, or number of transmissions). This ensures that even though the compute nodes in proximity may be bypassed because they are likely to be busy, their availability is examined frequently. The exact value of this parameter depends on the reliability and volatility of the system, i.e., fog computing systems that are expected to change frequently, should perform a reset more often than stable systems. This parameter may also be adaptive, and change dynamically based on the performance of the system. For example, if it is observed that when resetting the compute nodes component, a compute node in closer proximity is found, it may be beneficial to start resetting more regularly. On the other hand, if resetting the compute nodes component results in selecting the same node as before, then resetting should be performed less frequently. This way, the frequency of the reset can converge towards a suitable value.

Finally, we note the applicability of our approach for fog computing systems with multiple IoT devices, gateways, and compute nodes. As discussed in Section 3, in this paper we focus on the communication of one gateway (with multiple IoT devices and compute nodes), because other gateways of the system form similar paths to the cloud. Nevertheless, our system model and the proposed mechanism apply to systems with many gateways as well. Since each gateway of the system stores only few compute nodes and selects the most suitable among these nodes (rather than selecting one among all the nodes of the system), our mechanism follows a decentralized approach. In contrast to centralized approaches in which global view of the system is assumed for selecting the most suitable node, decentralized approaches tend to scale better while causing less overhead [50], [51].

4.3 Reinforcement Learning for the Data Analysis Component

In this section, we propose a concrete strategy for the data analysis component, which is based on RL [52]. Specifically, in Section 4.3.1 we adapt the RL model to the proposed context-aware mechanism, and in Section 4.3.2 we present the learning algorithm.

4.3.1 Reinforcement Learning Model

RL relies on agents that interact with their environment and receive feedback for their actions in the form of reward or penalty. Based on this feedback, the aim of the agents is to learn a policy in terms of selecting actions given the perceived state of the environment, so that the expected cumulative reward is maximized. In our case, the agent is installed at the gateway and at each time step, i.e., upon the arrival of an application request, selects the appropriate compute node to forward the request to, based on the environment's current state. Then, the agent inspects the latency experienced to handle this request, which constitutes the reward signal. This signal is considered as the feedback which is used in a process towards learning to select appropriate actions.

The agent has a set $\mathcal{A}$ of available admissible actions, with A_t denoting the action of the agent at time t. In our case, the actions available to the agent are the different compute nodes that can receive an application request from the gateway. Therefore, $A_t = i$ represents the action to forward the application request to node i at time t. Each action brings the environment to a potentially different state, out of a (finite and discrete, in our case) state space $\mathcal{S}$: When at state $S_t = s$, selecting action $A_t = a$ will lead to state $S_{t+1} = s'$ with a specific transition probability $p(s'|s, a)$, which is however unknown to the agent. Finally, the immediate reward the agent receives when performing action a and the system is at state s at time step t, is given by a reward function $R_t(s, a)$. Since our agent can only directly observe the latency experienced for each request, we use its normalized value as the reward, i.e., $R_t(s,a) = 1 - \frac{l_t(s,a)}{l_{max}}$, where $l_t(s,a)$ is the latency measured by the gateway for the t^{th} application request when from state s compute node a was selected, and l_{max} is a reasonably large maximum latency value.

In RL, the environment is typically modeled as a Markov Decision Process (MDP), defined as the 4-tuple $(\mathcal{S}, \mathcal{A}, p, R)$ [52]. The key characteristic of an MDP is the Markov property, namely that the probability to transit to a state depends only on the current state and the action taken. Otherwise put, a state should encode all the information necessary to drive an agent's decision, and the reward received should only depend on this state-action pair.

However, this assumption may not hold in some practical use cases for our system model. Importantly, the actual state of the environment might not be able to be fully observed. In our case, for example, the state in terms of the current load of each compute node, its reliability, and in general its probability to accept an application request, which, intuitively, are critical to drive the agent's decisions, are not visible by the agent. Instead, the agent is only aware of some environment observations, such as the actual node that responded to a request. We are therefore forced to operate only with an *approximate* view of the state of the environment. The agent however, can record a history of observations as a response of the environment to the agent's actions. Following the terminology and approach of Sutton and Barto [52, Section 17.3], what we need is a compact representation of a state as a summary of the history of observations and actions that have led the environment to a given state. The state thus becomes a *function of history*, i.e., $s_t = f(H_t)$, where H_t is the sequence of observations and actions up to time step t.

In our model, a state represents the history of how the k most recent requests were handled by the system. This is in line with what Sutton and Barto term as the k^{th} order history. In particular, $S_t = \{n_{t-k}, n_{t-k-1}, \cdots, n_{t-1}\}$, where n_i is the address of a compute node that processed a request; namely, the state of the system is maintained as an ordered list of the last k compute nodes that processed the respective application requests. In order to control the number of states, k is set to a small constant and can be configured based on the amount of memory available to the agent. Furthermore, this representation of state as a function of history only includes observations and not the actions that led to them. This is a simplification that was also driven by the need to keep the state space low.

Importantly, such a state representation is not guaranteed to come with the Markov property. For example, this may be the case if the probability of a node to accept a request depends on whether the node has recently been used for a prior request, as is the case for the compute node model that we assume in our evaluation (in Section 5.3). In this work, the agent is considered to be agnostic of the actual behavior of the compute nodes, and is forced to operate on minimal observed information regarding their status, namely the address of the compute node that accepted an application request, and the respective latency. The system in question may not be able to be represented as an MDP, due to how nodes operate, and because of the inability of an agent to fully observe the state of the environment. We apply Q-learning [53], which is a generic model-free RL mechanism, to derive the agent's policy (in Section 4.3.2). However, due to the above-mentioned fact, convergence guarantees cannot be provided for general settings. Interestingly, there are families of decision processes beyond MDPs for which Q-learning has been shown to come with convergence guarantees [54].

As a final note, we treat each context individually. Namely, there is a dedicated agent for each context which acts upon this particular context's application requests, and which maintains its own state of the environment (that is not shared across agents). This decision is motivated by the following observations:

- Whether a request for a specific application is served by a node is considered to be unaffected by whether this node would accept requests for other applications. For example, a node might have enough resources to serve a specific application request, but not enough for another.
- By assuming a different decision process (and, thus, a dedicated agent), per context, and given our state model, we limit the number of states handled by each agent to N^k, where N is the number of compute nodes and k is a small constant (representing the history). The total number of states of all M agents is then MN^k. Had we opted for factoring the application context in a single global state representation for all applications, e.g., where the elements of a state would be application instance-node address pairs, we would have come up with a single agent with $(MN)^k$ states, that is considerably larger.

4.3.2 Q-Learning Algorithm

We apply Q-learning [53] to learn an appropriate action selection policy in our system. A Q-learning algorithm aims at deriving a value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which represents the expected cumulative discounted future reward if the agent selects action a at state s, and continues by following the optimal policy. Q can be represented as a table, which is updated each time an agent takes an action, using the following rule:

$$\begin{aligned} Q_{t+1}(S_t, A_t) = {} & (1 - a_t(S_t, A_t))Q_t(S_t, A_t) + \\ & a_t(S_t, A_t)(R_t(S_t, A_t) + \gamma \text{max}_a Q_{t+1}(S_{t+1}, a)) \end{aligned}$$

where $\alpha_t(s,a) \in [0,1]$ is the learning rate at time step t and $\gamma \in [0,1]$ is a factor that is used to discount future rewards. The higher the value of γ, the higher the importance of future rewards compared with immediate ones.

As it is used at runtime and starts with an unknown Q function, the Q-learning algorithm needs to make decisions while learning. Therefore, the agent alternates between *exploration* and *exploitation* steps. There are various mechanisms to perform this. A widely used mechanism is the ε-greedy strategy, where the agent chooses a random action (thus exploring the environment) with probability ε, while with probability $1 - \varepsilon$ it follows the action with the highest Q value (thus exploiting the acquired knowledge of the environment). In order to favor exploration at the beginning, we start with $\varepsilon = 1$ and gradually reduce it as rounds progress. The learning rate for each state-action pair is initially set to 1 and also decays the more the pair is visited [55].

A point that requires further attention is that an implicit assumption is made that the state and action spaces are static, as is the case for the data structure which stores the Q values. However, according to the proposed mechanism described in Section 4.2, the compute nodes component which stores compute node addresses (these correspond to actions, while histories composed of these addresses correspond to states), starts with only the address of a single node in proximity. When the addresses of the compute nodes on path are not known in advance, potential methods to get around this can be applied. Such methods can be:

1) To start with a discovery phase whereby traditional routing is executed for a sufficient number of rounds until the nodes on path are discovered [56], and then to switch to Q-learning. The duration of the discovery phase depends on the load of application requests, and the number of nodes that accept these requests. For example, when application requests are sent sparsely, the discovery phase may take longer than if application requests are sent frequently. Also, the fewer the compute nodes which accept the requests, the faster the discovery phase finishes. For these reasons, this method is preferred when the gateway produces a high load of application requests and/or the number of nodes that accept these requests is bound.
2) To build the Q table incrementally. This means adding new states and admissible actions as new nodes are being discovered, each time maintaining the existing Q values for the already known state-action pairs and learning the values of new ones by appropriately adjusting the exploration and learning rate parameters. Thus, this method requires additional updating of the Q table every time a new node is discovered. When using this method, the acquired knowledge is utilized from the beginning because there is no discovery phase (in contrast to the former method). Nevertheless, the additional updating of the Q table might introduce extra overhead. Hence, this method is preferred when new nodes are expected to be discovered sparingly.

Since these methods are executed until sufficient knowledge is acquired (i.e., not constantly), in our evaluation in Section 5, we assume that the compute nodes on path are known (e.g., due to a discovery phase).

It is worth noting that the proposed mechanism operates on minimal context and feedback: *i*) the latency observed per request, and *ii*) the compute nodes that accepted the request. For this reason, our mechanism does not utilize an excessive amount of resources (this is also discussed in Section 5.3.3). Each time a request is handled, the overhead of selecting the best action and updating the Q table depends on the number of possible actions. The actions correspond to the stored compute nodes, i.e., the nodes that accept application requests, which are only a subset of the compute nodes of the system. Thus, by storing only a subset of the compute nodes of the system, the number of actions remains low which limits the computational overhead (and latency) of handling each request. Our model and mechanism are also agnostic to the behavior and actual state of the compute nodes on path. If we assume further knowledge about the environment (e.g., information about the runtime of each compute node such as: the current load, device capabilities, probability to accept new application requests, etc.), a more accurate view of the state can be acquired, and more efficient mechanisms may be possible. However, this could potentially increase the utilization of computational resources. We defer such mechanisms, as well as other approaches for dealing with an uncertain environment, to future work.

5 Evaluation

In this section, we present an implementation of the proposed context-aware routing mechanism. Furthermore, we conduct a series of experiments, and we compare the proposed context-aware routing to a state-of-the-art routing approach which we use as baseline. This baseline is based on [7] and [9], which propose that each compute node on path examines an application request and either accepts it or forwards it to the next node. Hereinafter, we refer to this approach, which is also shown in Fig. 3a, as *traditional routing*.

In order to show the differences compared to traditional routing, we evaluate the proposed mechanism in the following manner: First, we establish the potential benefits of performing context-aware routing on a real-world setup with nearby and remote compute nodes, and we show that even a simplistic strategy for the data analysis component has the potential to outperform the traditional routing approach. Then, we turn our attention to the performance of the RL-based strategy, and we conduct extensive simulations that show benefits in various scenarios.

The required files to reproduce our experiments including source code and executable files, as well as the produced numerical results of our evaluation, are available in the online repository [57].

5.1 Evaluation Environment and Prototype

To create an evaluation environment, we implement a prototype of a compute node which either accepts an application request or forwards it to the next node on path, as discussed in Section 3. Moreover, we implement a prototype of a gateway which integrates the proposed mechanism for context-aware routing, as discussed in Section 4.2. Both prototypes are developed in Java 11 using the Spring Framework, and implement the required functionality to perform experiments and take measurements related to the communication latency and hop count of the utilized network paths.

To emulate a fog computing system, we assume the following scenario. An IoT application provider based in Los Angeles has successfully commercialized smart energy solutions (e.g., for detecting gas leaks, reducing cost, etc.). This provider uses cloud computing resources stationed in a data center at Los Angeles, but due to the popularity of the provided IoT applications, the provider decided to expand this business to the rest of the US and to Europe.

Since such IoT applications can be related to safety (e.g., due to detecting gas leaks), and may require low latency that the centralized cloud might not be able to support [58], the provider decided to follow the fog computing paradigm. Thus, the provider acquired access to various geographically distributed compute nodes in the cloud (offered by cloud provider such as Google or Microsoft), and at the edge of the network (e.g., using the edge zones offered by Microsoft [31]).

For this evaluation, we examine the case of a client in central Europe (i.e., in Vienna, Austria), and we consider that various compute nodes exist on the path towards the cloud in Los Angeles, as shown in Fig. 5. Notably, we do not distinguish between edge and cloud compute nodes because: *i*) Both cloud and edge nodes are able to provide the same services [31], and can therefore be considered similar. *ii*) In this setup, we use nearby and remote compute nodes using cloud services because even though edge nodes have been announced by cloud providers [31], their availability is still limited. Thus, similar to Fig. 1 which shows our system model, we examine a fog computing system with a gateway which sends application requests that are forwarded by various compute nodes towards the cloud. The specific compute nodes we use (as shown in Fig. 5), are provisioned using the Google Cloud Platform. The type of the compute nodes is *e2-standard-2*, i.e., standard general-purpose compute nodes with 2 vCPU and 8 Gigabytes of RAM (although the resource capacities of the compute nodes do not affect the presented results significantly because this evaluation focuses on network-related metrics). Hence, for this evaluation we create an actual computing system with nearby and remote compute nodes that span a large geographical area. Interestingly, the remote compute nodes in the US can provide insights on the communication over high-latency network links. This is useful because high-latency links represent cases when remote compute nodes are employed due to the nearby nodes being busy. In addition, high-latency links may be considered representative of scenarios that include network links which induce high latency for other reasons, e.g., due to congestion.

In this system, the gateway receives gas volume measurements from a smart meter. Then, the gateway creates the application request of an application that detects gas leaks. In the event that a gas leak is detected, the application sends an actuation command to shut down all sources of ignition, e.g., cooktops, toasters, etc. Therefore, for such an application, reducing the communication latency of sending the gas measurements to a compute node, aids in processing the data faster, and reduces the overall latency of responding to fire hazards. For this reason, we apply the proposed approach which aims at reducing the communication latency of sending data to nearby and remote compute nodes, in order to examine the potential benefits.

To emulate the smart meter, we use real gas volume measurements from a smart home, which have been collected periodically every 30 minutes during the course of 4 days, i.e., 200 measurements. This dataset is part of the data provided by the Loughborough University, which has been gathered in the context of the REFIT project that monitored 20 smart homes in the United Kingdom [59].

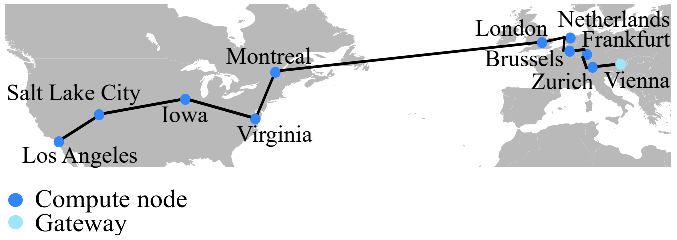


Fig. 5: Location of the compute nodes used in the evaluation.

5.2 Prototype-based Results

In this section, we use our prototype implementation to create fog computing systems based on the traditional and context-aware routing approaches, and we report on our findings. Specifically, Section 5.2.1 presents comprehensive results of the communication latency to reach every compute node of the system based on each routing approach. For these results, we assume that the data analysis component of the context-aware routing is able to determine the compute node that accepts the application requests. Afterwards in Section 5.2.2, we present results from the runtime of a specified scenario with dynamic load from application requests. In addition, the data analysis component utilizes a rather simplistic strategy: Send each application request to the most recent node that has accepted requests of the same context, and reset the compute nodes component with every second transmission. This experiment aims at showing that even in cases which do not allow for great benefits (e.g., with dynamic load and a very simplistic data analysis component), the proposed context-aware routing mechanism may still be able to show improvements.

Since the proposed approach aims at reducing the communication latency (which is independent of potential processing delays), this evaluation reflects on that by focusing on network-related metrics (rather than application execution metrics). Nevertheless, the additional utilization of computational resources in the gateway, which is required to execute the context-aware routing, is considered as the overhead of the proposed approach. For this reason, we discuss resource utilization aspects in Section 5.2.3. For all the presented results, we have repeated each experiment 200 times with the values of our dataset in order to capture the general behavior of each examined approach.

5.2.1 Context-Aware Routing Results

In order to acquire a comprehensive view of the two examined approaches considering that context-aware routing is able to determine the node that accepts each application request, we perform the following experiment. The gateway sends application requests to each compute node of the system two times. One time the data is sent according to traditional routing (i.e., on a path towards the cloud), and the other time according to the context-aware routing (i.e., directly to the selected compute node). Each time, we measure the communication latency to reach the compute node, and the hop count (using Traceroute which is a command line tool for network diagnostics). In the communication latency, we also include message parsing and making the data available to the application hosted in the destination

TABLE 1: Communication latency (in ms) and hop count to reach each compute node of the system based on the two examined approaches, and the percentages of reduction.

	Traditional routing		Context-aware routing		Reduction (%)	
	Hop count	Latency (average / st. dev.)	Hop count	Latency (average / st. dev.)	Hop count	Latency (average)
1. Zurich	15	93 / 11	15	93 / 7	0	0
2. Frankfurt	21	104 / 12	18	96 / 22	14	8
3. Belgium	30	118 / 20	22	100 / 11	27	15
4. Netherlands	43	127 / 16	23	101 / 17	47	20
5. London	53	140 / 10	18	108 / 5	66	23
6. Montreal	58	264 / 21	21	224 / 18	64	15
7. Virginia	65	282 / 14	23	226 / 17	65	20
8. Iowa	80	321 / 16	22	253 / 18	73	21
9. Salt Lake City	89	362 / 21	25	283 / 4	72	22
10. Los Angeles	95	385 / 13	26	296 / 4	73	23

node (but exclude any further processing by that application). Table 1 shows these measurements for each compute node, along with the percentages of reduction when using the proposed context-aware routing approach.

Notably, the hop count to reach each node of the system is always the same in the 200 iterations, because the underlying network connectivity is not affected by our experiments. For this reason, in Table 1 we do not specify average and standard deviation of the hop count. The reason we measure the hop count is because network hops can be used as an indicator of bandwidth availability since nodes that reside many hops away from each other, tend to communicate with low bandwidth [60]. Therefore, lower hop count can be associated with higher bandwidth [14], [61].

In traditional routing, whereby each application request is forwarded to the compute node in closest proximity on a path towards the cloud (as shown in Fig. 5), the average latency increases the farther away from the gateway the compute node is located. In the context-aware routing, whereby the gateway sends the application request to each compute node directly, the average latency also increases the farther away the compute node is located. However, the rate of increase is significantly lower in the context-aware routing.

Both approaches reach the first compute node with 15 hops in 93 milliseconds (ms) because both approaches send the application request to the closest compute node directly. After that, the difference in both the communication latency and the hop count becomes evident. Specifically, the context-aware routing approach reduces the average communication latency by up to 23%, and the average hop count by up to 73%. Notably, even though the difference in the communication latency becomes larger when the distance of the compute node becomes longer, the percentages of reduction become significant even with compute nodes that reside nearby. For example, to reach the compute nodes in the Netherlands and in London (from Vienna), the context-aware routing reduces the average communication latency by 20% and 23%, respectively. This shows that our approach shows benefits for both inter- and intra-continent communication. The former can be representative of large-scale applications that operate wordwide, while the latter may be considered indicative of results for medium-scale systems (e.g., that operate within Europe).

To further analyze the rate of increasing communication latency for each routing approach, we plot the average values of Table 1 in Fig. 6. Since the values shown in this figure exhibit the pattern of straight lines, we also show the linear trendlines which are created using the Least Squares method. Even though the communication latency increases monotonically (which is expected since the physical distance of the compute nodes increases), there is a steep rise in the latency of the compute node 6, i.e., the node in Montreal. This derives from the increased latency to reach the remote compute nodes in America, due to the significantly longer distance from the gateway (which can also be observed in Fig. 5). Notably, this rise prevents the values of each routing approach from following the pattern of a straight line holistically. However, it is visually evident that the values of the group of nodes before the rise (i.e., the latency of the nodes in Europe), and the values of the group of nodes after the rise (i.e., the latency of the nodes in America), both exhibit a linear pattern. For this reason, Fig. 6 shows a different trendline for each group of nodes.

For each trendline, we also present its linear function in the form of $y = Slope \cdot x + Intercept$. We do this because the slope of a linear function shows the rate of increase of the output values. Thus, since the y values of Fig. 6 show communication latency, and the x values show the nodes in the order of increasing distance, the slope indicates the rate of increased latency based on the proximity of the gateway (for each routing approach). In addition, we present the coefficient of determination R^2 for each trendline. R^2 is a widely-used statistical measure which indicates how close are the actual values to the values of the trendline [27]. The value of R^2 is always between 0 and 1 [62]. Small values indicate that the model (or the trendline in our case) does not represent the data well, while high values show that the model can be considered representative for the data [63]. Thus, we use R^2 to advocate that the presented trendlines, and consequently the slopes which indicate the rate of increase in the communication latency, bear statistical significance.

For the compute nodes in Europe, the function of the trendline of the traditional approach is $y = 11.7 \cdot x + 81.3$, and has a coefficient of determination $\mathrm{R}^2 = 0.997$. This coefficient of determination shows that the trendline represents the data very well. In addition, we note that the slope of this trendline is 11.7. For the same nodes, the function of the trendline of the proposed approach is $y = 3.5 \cdot x + 89.1$, and has a coefficient of determination $R^2 = 0.948$. While the coefficient of determination of this trendline is also very high, the slope is significantly lower, i.e., 3.5. This means that in the context-aware routing, the communication latency grows by a factor of 3.5 which is about 70% lower than in the traditional routing.

For the compute nodes in America, we note that the function of the trendline of the traditional approach is $y = 32.2 \cdot x + 65.2$, and has a coefficient of determination $R^2 = 0.985$ which shows that the trendline exhibits statistical significance. We also note that the value of the slope of this trendline is 32.2. For the same nodes, the function of the trendline of the proposed approach is $y = 20.1 \cdot x + 95.6$, and has a coefficient of determination $R^2 = 0.948$. While R^2 is also very high in the proposed approach, the slope of the trendline is much lower, i.e., 20.1. Specifically, this slope shows that the communication latency in the context-aware routing grows by a factor of 20.1 which is about 38% lower than in the traditional routing.

Therefore, we conclude that the communication latency of sending IoT data to compute nodes of fog computing systems increases for both routing approaches, when the nodes reside farther away from the gateway. However, the rate of increase is significantly lower in the context-aware routing approach. As a result, context-aware routing is able to send the IoT data with lower communication latency than the traditional approach.

5.2.2 *Results From a Scenario with Dynamic Load*

In this section, we aim at showing that the proposed context-aware routing mechanism may be able to outperform the traditional routing approach, even in cases which do not allow for great benefits. To this end, we measure the communication latency of the context-aware routing when using a very simplistic strategy in the data analysis component. In the following, first we describe the details of the scenario for this experiment, and then we present the produced results.

Scenario: The strategy we use in the data analysis component of the context-aware approach is the following: First, send an application request using the traditional approach, and store the compute node that accepted this request. Then, send the next request directly to the node that accepted the data before. After that, the compute nodes component is reset and the process is repeated.

In addition, we consider the potential load from other application requests and other gateways, along with the potentially limited resources of the compute nodes on path. This is emulated in our system by configuring the compute nodes on path to have 7% probability of accepting an application request, while the final cloud compute node has 100% probability (due to integrating virtually unlimited resources, as discussed in Section 3). Since there are 9 compute nodes (apart from the final cloud node), the probability of all these nodes rejecting an application request (and thus the request reaching the final cloud node) is $(1 - 0.07)^9 = 0.52$, i.e., approximately 50%. We do this to emulate a realistic environment for our experiments (as discussed in Section 4.1), whereby approximately half of the application requests are executed by nodes on path, and the other half in the cloud.

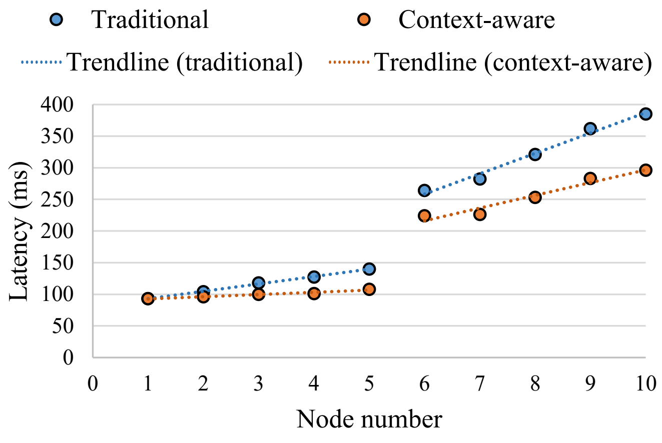


Fig. 6: Average communication latency to reach each compute node of the system based on the two examined routing approaches (values acquired from Table 1).

After executing an application, the probability of accepting another application request increases from 7% to 50% for every compute node. This is done to emulate that when one of the hosted applications finishes, a compute node has more available computational resources. This increase is temporary, and lasts until the next application request is sent from the gateway (after that the probability is reset to 7%). The reason that the probability increase is temporary, is to emulate that other gateways in the area may send application requests, and occupy the newly available computational resources.

Results: For this experiment, the gateway sends each one of the 200 application requests two times according to the traditional and the context-aware routing approaches. For the context-aware, the gateway uses the mechanism discussed in Section 4.2, and the aforementioned strategy for the data analysis component. The produced results which are related to the communication latency and the hop count of the network paths of each routing approach, are shown in Fig. 7. Similar to the values of Table 1, for the communication latency we include message parsing but exclude any further processing by the application in the destination node.

Fig. 7a shows the distribution of the hop count of the application requests for both approaches. The traditional routing measurements have an average value of 72 hops, and a standard deviation of 33 hops. The context-aware routing measurements have an average value of 51 hops, and a standard deviation of 32 hops. Thus, the proposed approach reduces the average hop count by 29%. Notably, the box plot of the traditional routing does not have an upper whisker, and the median overlaps with the upper quartile. The former indicates that 25% of the maximum values are equal, and the later that 50% of the maximum values are equal.

This happens because approximately half of the application requests are sent to the final cloud node which bears the maximum hop count. Thus, as Fig. 7a indicates, half of the values of the hop count in traditional routing equal the maximum value. The same applies to the context-aware routing, i.e., approximately half of the application requests are sent to the final cloud node. However, since the context-aware routing sends some of these requests to that node directly, the hop count is significantly lower (as shown in Table 1). For this reason, the box plot of the hop count in the context-aware routing has lower values, and does not exhibit similar behavior.

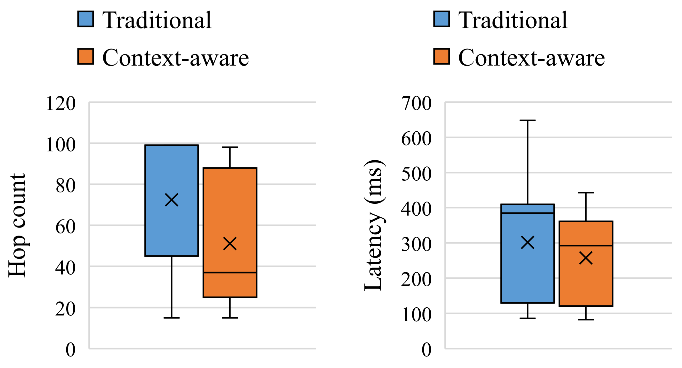
In addition, we note that the maximum hop count in this experiment, which represents the number of hops to reach the cloud node in Los Angeles, is 99 hops. This is slightly different to the hop count of Los Angeles shown in Table 1, which is 95 hops. The reason that this happens, is that we use the external IP addresses for the communication between compute nodes. The Google Cloud Platform allows communication between Google compute nodes, using either internal IPs (that can be used only between Google nodes), or external IPs that are addressable by any node on the Internet. The network paths when using internal IPs may be more stable, which could prevent having slightly different hop count between experiments. However, a setup that utilizes external IPs, is more representative of computing systems which consist of compute nodes that may belong to different providers (which is possible in fog computing systems). Nevertheless, when using external IPs, slightly different hop count values between experiments might be observed, due to the dynamic nature of the Internet. We do not consider that this compromises the presented results. On the contrary, we believe that results which include slight dynamic changes are more representative of real-world setups, in which, such phenomena are imminent.

Fig. 7b shows the distribution of the communication latency. The measurements of the traditional routing have an average value of 302 ms, and a standard deviation of 140 ms. The measurements of the context-aware routing have an average value of 258 ms and a standard deviation of 113 ms. Therefore, the context-aware routing reduces the average latency by 15%. Notably, the median of the traditional routing, is higher than the upper quartile of the context-aware routing. This means that 75% of the values of the context-aware routing are lower than 50% of the values of the traditional routing. This further advocates that the proposed context-aware routing mechanism is able to reduce the communication latency of sending application requests in fog computing systems.

5.2.3 *Resource Utilization*

As discussed in Sections 5.2.1 and 5.2.2, context-aware routing is able to provide latency and bandwidth benefits due to using the mechanism of Section 4.2. The implementation of this mechanism however, may require the utilization of additional computational resources in the gateway, which can be regarded as the overhead that accompanies the presented benefits.

In our experiments, the gateway is implemented on a Raspberry Pi 4 single-board computer (which is a popular choice for an IoT gateway [35]). Using Top (a command line tool for monitoring resource utilization), we examine the resource utilization of the process that routes the application requests according to each routing approach. Notably, both approaches exhibit similar resource utilization with CPU that has occasional spikes which do not surpass 9%, and



(a) The hop count of the application requests.

(b) The latency of the application requests.

Fig. 7: Network path measurements of the application requests based on traditional and context-aware routing.

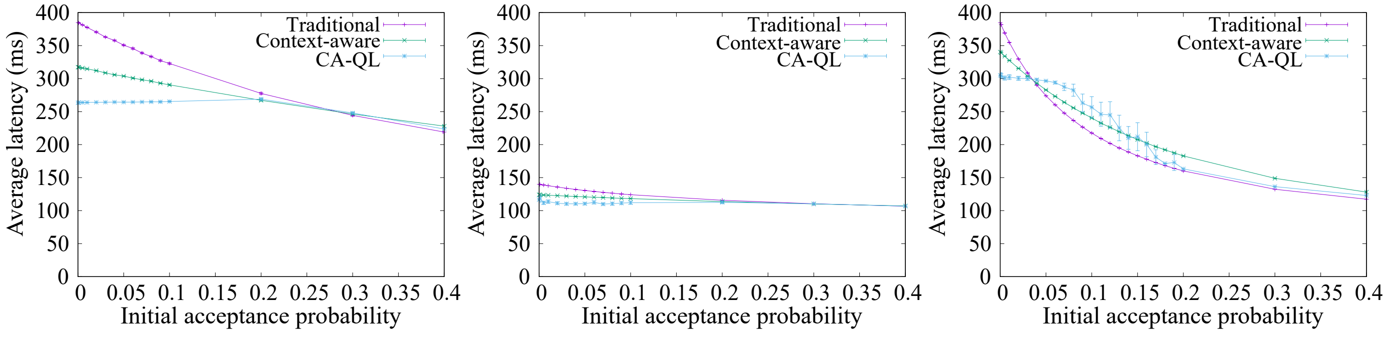
RAM that does not exceed 10%. The resource utilization of the compute nodes which execute the IoT applications, is not affected by our approach since the context-aware routing mechanism is only executed in the gateway.

In addition, we note that the time required to execute the proposed mechanism in our experiments, i.e., the delay needed for finding a compute node that has accepted data of certain context in the past, is negligible. However, this delay may increase if a complex data analysis component is used, or if the number of different contexts stored in the compute nodes component grows significantly (due to the delay of searching). Therefore, the presented results apply to fog computing systems with a gateway which makes application requests of a reasonable number of different contexts (i.e., when a search in the compute nodes component does not incur significant delay).

The application responses used in the proposed approach might also be considered as overhead since the traditional routing does not make use of such information. Nevertheless, when a reliable communication protocol is employed for the communication between compute nodes (e.g., using the request/response HTTP which we use in our experiments), the address of a compute node can be encapsulated within the HTTP responses. This way, the application response can be sent without transmitting additional messages to the gateway. Thus, we conclude that the context-aware routing approach using a rather simple data analysis strategy, is able to provide solid benefits without inducing significant additional overhead.

5.3 Simulation-based Evaluation

In this section, we turn our attention to the performance of our mechanism when using the RL-based strategy for the data analysis component (presented in Section 4.3) under a wide range of settings and scenarios. To evaluate this strategy in a comprehensive manner, we build a Python-based simulator which allows us to increase the scale of our experiments substantially (compared to using the setup of Section 5.1). In addition, we model our simulator to consider the network parameters of our real-world setup (discussed in Section 5.1) in order to ensure that the simulations imitate



(a) Small scale. (b) Medium scale. (c) Large scale.

Fig. 8: Average communication latency achieved by the three examined routing approaches for each deployment scenario, as the probability that a compute node accepts an application request grows.

a realistic environment. The results we report in this section are based on the average latency achieved from 100,000 time steps, i.e., for the first 100,000 application requests. Each experiment is repeated 20 times, and the average values are reported with 95% confidence intervals.

When running our experiments, we compare three routing approaches: *i*) The traditional routing approach. *ii*) The simple context-aware routing approach discussed in Section 5.2. *iii*) The RL-based approach with Q-learning discussed in Section 4.3, which is referred to as Context-Aware with Q-Learning (CA-QL).

5.3.1 Scenarios

To examine the performance of the three routing approaches, our simulator considers three deployment scenarios with inherently different characteristics, as shown below:

- **Small-scale deployment**: In this scenario, there are two compute nodes: a local edge node which is assumed to be very close to the gateway (average latency of 10 ms), and a remote cloud node which is located far away (with average latency corresponding to the node in Los Angeles, as shown in Table 1). This can be considered representative of small-scale setups with one nearby edge node, and one remote cloud node. Notably, this scenario resembles setups which have transitioned from the cloud computing paradigm by adding a local edge node.
- **Medium-scale deployment:** This scenario includes a chain of 5 compute nodes deployed in the same continent. The communication of these compute nodes corresponds to the nodes in Europe (from Section 5.1), and is based on the latency values of Table 1. This scenario is considered representative for setups of medium-scale fog computing systems which operate in a wide area within the same continent.
- **Large-scale deployment:** This scenario corresponds to the real-world setup discussed in Section 5.1, and includes a chain of 10 compute nodes which communicate with the average latency values reported in Table 1. We consider this as a representative scenario for large-scale setups of fog computing systems which may need to operate worldwide.

In addition, we experiment with different cases of nodes accepting new application requests, in each of the three target deployment scenarios. In particular, we consider that each compute node accepts an application request with a probability, as discussed in Section 5.2.2. However, in this experiment, we vary this probability in order to acquire results which cover a wider range of fog computing systems. After a compute node has accepted a request, the probability of accepting the subsequent one increases to a higher value (i.e., to 50%, as discussed in Section 5.2.2). In line with our prototype-based experiments, this applies to all compute nodes but the final cloud node, which always accepts new application requests.

We do this because the main factor that determines the performance of our context-aware routing approach, is whether the nearby compute nodes have sufficient available resources to accept new application requests. In case these nodes are busy, our approach may reduce the communication latency by sending the application requests directly to nodes with available resources. Thus, by considering different application request acceptance probabilities, and different deployment scenarios, this experiment aims at verifying that our context-aware routing mechanism is able provide latency benefits in fog computing systems, for a wide range of parameters.

5.3.2 Results

In Fig. 8, we show the results of our simulations using the three routing approaches, and considering the three target deployment scenarios. In CA-QL, the discount factor is set to $\gamma = 0.9$ which is a common value for γ, and is usually selected due to reducing the long-term penalty [64].

In each deployment scenario of Fig. 8, we can observe that all routing approaches tend to converge towards the same minimum latency value when the initial acceptance probability increases. This happens because when this probability becomes high, it is the first compute node on path that accepts most of the application requests. Thus, the context-aware routing approaches are not able to provide benefits by bypassing busy nodes, since most nodes tend to have enough spare resources to accept new requests. This observation shows that when the proposed approach is not able to provide benefits, the performance of the system does not degrade significantly, but rather produces similar results to the traditional approach.

In our target environments, such as IoT systems that process huge amounts of data, the probability of nearby nodes accepting new application requests is rather low, as discussed in Section 4.1. For this reason, we consider the results with low initial acceptance probability more relevant for this evaluation. According to these results, in Fig. 8a which shows the average latency in small-scale deployments, we note that the context-aware approaches, i.e., the simple context-aware and the CA-QL, both outperform the traditional routing consistently, while the RL-based CA-QL also outperforms the simple context-aware. Similar behavior can also be observed in Fig. 8b which shows the average latency of medium-scale deployments. Notably, the latency reduction of CA-QL compared to the traditional approach (in Fig. 8b), reaches 19.5%. Based on the latency values of the nodes in Europe which are used in this experiment, this reduction approximates the maximum possible reduction in the average latency (shown in Table 1). This indicates that the results of Table 1, which are produced assuming that the proposed context-aware mechanism is able to select the most appropriate compute node, are actually achievable when implementing predictive methods (such as the CA-QL) in the data analysis component.

Finally, for Fig. 8c which shows the average latency of large-scale deployments, we note that the context-aware approaches show benefits for a range of small probabilities, but when the probability increases, the results are mixed. This happens due to the high-latency to reach the remote compute nodes in another continent. When most of the application requests are accepted by the final cloud node (i.e., small acceptance probabilities), the traditional approach has the highest latency due to the forwarding by the nodes on path. In this case, the context-aware approaches provide the benefits shown in the left-hand side of Fig. 8c until the intersection point (at approximately 4% probability), due to sending some of the application requests directly to the final cloud node. Notably, when an application request does not reach the cloud, but is accepted by a remote compute node (i.e., in another continent), the context-aware mechanism favors that remote node for the subsequent request even though the latency may be high. The traditional routing on the other hand, does not favor any nodes, which means that the subsequent request is sent on a path and might be accepted by a nearby node. For this reason, when there are compute nodes in different continents, and the acceptance probability increases, the traditional approach tends to have lower latency than the context-aware. Hence, after the intersection point (in Fig. 8c), the traditional routing performs better.

Notably, similar behavior may also occur before the intersection point. However, the benefit from sending some of the application requests to the cloud directly, which are achieved by the context-aware approaches, are large enough to overshadow the shortcomings. This is also the reason that our prototype shows the benefits of Fig. 7b using 7% probabilities. The results of Fig. 8c are not completely in alignment with Fig. 7b because the latency between the nodes in our simulations, is based on Table 1 which has slightly different values than the values of Fig. 7b (due to using actual Internet measurements and external IP addresses, as discussed in Section 5.2.2). Nevertheless, while Fig. 7b shows that our prototype can provide benefits in a large-scale deployment, Fig. 8c shows that there is actually a range of small probabilities for which the context-aware approaches can provide benefits. Outside this range, our simulations show that the traditional approach is slightly better than the simple context-aware, while CA-QL exhibits high variance, but does not perform better than the traditional.

Discussion: In general, we observe that the exact latency achieved by each routing approach is a function of the specific environment, i.e., the latencies in the underlying network, and the node acceptance probabilities. Nevertheless, we note that for low acceptance probabilities (which is our target environment), the context-aware approaches consistently outperform the traditional routing in all our experiments (both prototype- and simulation-based). This happens because the traditional approach takes time to send the application requests on a path of compute nodes towards the cloud, until a node eventually accepts it. The simple context-aware routing, on the other hand, takes advantage of the awareness of the last node that accepted the request, which may be a node farther along the path, and bypasses all previous nodes, thereby saving time. CA-QL performs even better than the simple context-aware in most settings we examine, with performance gains that are more pronounced in the small- and medium-scale deployment scenarios. This occurs because CA-QL learns to direct application requests to cloud nodes when other intermediate compute nodes are likely to be busy. Notably, it can be observed that CA-QL strikes a good balance across the range of parameters explored: When it pays off to bypass intermediate nodes, it usually performs better than the alternatives, while as soon as intermediate nodes become more eager to accept requests, its performance converges to that of the traditional routing.

When the compute node acceptance probabilities are low, the gains of CA-QL over the traditional approach in terms of latency reach up to 31% and 19.5%, for the small- and medium-scale deployment scenarios, respectively. Even though in the large-scale scenario, we observe a range of parameters for which the simple context-aware and the traditional routing perform better, we note that CA-QL has more tangible benefits overall. Notably, in our experiments we assume that the exact behavior of the compute nodes, is not known in advance (e.g., through monitoring the available resources). Therefore, even if it may not be entirely possible to create a mechanism that always selects the closest compute node with available resources, we consider that CA-QL offers a very good compromise.

5.3.3 Memory Consumption

An important aspect of CA-QL is the amount of maintained state. As discussed in Section 4.3.1, for every stored context, an agent needs $O(N^{k+1})$ space to store state information, where N is the number of nodes and k is the number of the most recent responses used in our state representation (history size). The Q-table has N^k states, and for each state there are N available actions. The value of k can be selected with space limitations in mind. A large k may make it infeasible to operate the agent, especially on a resource-limited device, such as a Raspberry Pi which could be used as the gateway. Performance-wise, in the small- and medium-scale deployment scenarios, CA-QL is not very sensitive to the choice of k. However, in the large-scale scenario, low values of k (e.g., $k = 1$) might result in increased latency. For this reason, larger values may be more desirable. To appropriately tune the state space size given a specific maximum memory threshold M to be used by an application's Q-learning agent at the gateway, k can be set to $k \leq \lfloor log_N(M/c) - 1 \rfloor$, where c is the space necessary per state-action pair. In fact, this is the method we use to tune k in the experiments of Section 5.3.2, setting $M = 256\,\text{MiB}$. Finally, we should recall that for each different context, the gateway operates a separate Q-learning agent. This is something the system operator needs to keep in mind in order to configure appropriately the amount of memory resources dedicated to each agent process. In our implementation, $c = 8\,\text{bytes}$: one 32-bit floating point number to store the current estimate of the Q function value, and a 32-bit integer to store the counter of how many times the action has been selected given that state.

6 Conclusion

In this paper, we propose a routing mechanism which considers the context of the IoT data in order to route application requests to compute nodes of fog computing systems. To achieve that, a history of previous transmissions is kept, so that new application requests can be sent directly to compute nodes which usually accept requests of the same context. We evaluate this approach using a prototype implementation with distributed nearby and remote compute nodes, and by performing extensive simulations. The results from our prototype show that compared to a state-of-the-art method, context-aware routing reduces the latency of sending IoT data to compute nodes of fog computing systems by up to 23%, and lowers the hop count by up to 73% (which indicates lower bandwidth utilization). Furthermore, we show that when using a simple strategy for selecting compute nodes, our approach is able to reduce the latency and the hop count of the transmissions, without inducing significant overhead. In addition, we perform simulations on a wealth of settings, and we show sound benefits when combining the proposed context-aware mechanism with a RL-based predictive method. Based on these results, we deduce that such a routing mechanism can further advance fog computing systems.

Due to the promising results, in the future we plan to analyze further the information which is stored by our mechanism, i.e., the compute nodes, and the context that each compute node accepts. Specifically, we consider two promising research directions for future work: *i*) To explore effective ways for profiling the IoT data, and for extracting context. *ii*) To investigate additional ways for predicting the compute node in closest proximity which may accept new application requests (e.g., by examining/combining various machine learning models).

Acknowledgments

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

The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association is gratefully acknowledged.

References

[1] H.-Y. Wu and C.-R. Lee, "Energy efficient scheduling for heterogeneous fog computing architectures," in *Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, pp. 555–560, IEEE, 2018.

[2] C. Anglano, M. Canonico, and M. Guazzone, "Online user-driven task scheduling for femtoclouds," in *International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 5–12, IEEE, 2019.

[3] R. Ding, X. Li, X. Liu, and J. Xu, "A cost-effective time-constrained multi-workflow scheduling strategy in fog computing," in *International Conference on Service-Oriented Computing (ICSOC)*, pp. 194–207, Springer, 2018.

[4] B. Javadi, Q. L. Trieu, K. M. Matawie, and R. N. Calheiros, "Smart food scanner system based on mobile edge computing," in *International Conference on Cloud Engineering (IC2E)*, pp. 20–27, IEEE, 2020.

[5] 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 *Annual ACM Symposium on Applied Computing (SAC)*, pp. 751–760, 2018.

[6] M. Amadeo, A. Giordano, C. Mastroianni, and A. Molinaro, "On the integration of information centric networking and fog computing for smart home services," in *The Internet of Things for Smart Urban Ecosystems*, pp. 75–93, Springer, 2019.

[7] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 41–48, IEEE, 2017.

[8] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Annual IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1–9, IEEE, 2016.

[9] S. H. Mortazavi, M. Salehe, C. S. Gomes, C. Phillips, and E. de Lara, "Cloudpath: A multi-tier cloud computing framework," in *Symposium on Edge Computing (SEC)*, pp. 1–13, ACM, 2017.

[10] E. Kristiani, C.-T. Yang, C.-Y. Huang, P.-C. Ko, and H. Fathoni, "On Construction of Sensors, Edge, and Cloud (iSEC) Framework for Smart System Integration and Applications," *IEEE Internet of Things Journal*, vol. 8, pp. 309–319, 2020.

[11] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. S. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 939–951, 2019.

[12] B. Li, Q. He, G. Cui, X. Xia, F. Chen, H. Jin, and Y. Yang, "Read: robustness-oriented edge application deployment in edge computing environment," *IEEE Transactions on Services Computing*, 2020.

[13] G. L. Stavrinides and H. D. Karatza, "Cost-effective utilization of complementary cloud resources for the scheduling of real-time workflow applications in a fog environment," in *International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 1–8, IEEE, 2019.

[14] V. Karagiannis and S. Schulte, "Comparison of alternative architectures in fog computing," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2020.

[15] D. A. Chekired, L. Khoukhi, and H. T. Mouftah, "Industrial IoT data scheduling based on hierarchical fog computing: a key for enabling smart factory," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4590–4602, 2018.

[16] R. Mahmud, A. N. Toosi, K. Rao, and R. Buyya, "Context-aware Placement of Industry 4.0 Applications in Fog Computing Environments," *IEEE Transactions on Industrial Informatics*, vol. 16, pp. 7004–7013, 2019.

[17] T. Mononen, M. M. Aref, and J. Mattila, "Filtering scheme for context-aware fog computing in cyber-physical systems," in *International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, pp. 1–7, IEEE, 2018.

[18] D. S. Roy, R. K. Behera, K. H. K. Reddy, and R. Buyya, "A context-aware fog enabled scheme for real-time cross-vertical IoT applications," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2400–2412, 2018.

[19] A. Akbar, F. Carrez, K. Moessner, J. Sancho, and J. Rico, "Context-aware stream processing for distributed IoT applications," in *World Forum on Internet of Things (WF-IoT)*, pp. 663–668, IEEE, 2015.

[20] P. Wiener, P. Zehnder, and D. Riemer, "Towards context-aware and dynamic management of stream processing pipelines for fog computing," in *International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–6, IEEE, 2019.

[21] S. Basu, M. Karuppiah, K. Selvakumar, K.-C. Li, S. H. Islam, M. M. Hassan, and M. Z. A. Bhuiyan, "An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment," *Future Generation Computer Systems*, vol. 88, pp. 254–261, 2018.

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

[23] R. Casadei, D. Pianini, M. Viroli, and A. Natali, "Self-organising coordination regions: a pattern for edge computing," in *International Conference on Coordination Languages and Models*, pp. 182–199, Springer, 2019.

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

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

[26] S. Forti, M. Gaglianese, and A. Brogi, "Lightweight self-organising distributed monitoring of fog infrastructures," *Future Generation Computer Systems*, vol. 114, pp. 605–618, 2021.

[27] V. Karagiannis and S. Schulte, "Distributed algorithms based on proximity for self-organizing fog computing systems," *Pervasive and Mobile Computing*, vol. 71, p. 101316, 2021.

[28] "OpenFog Reference Architecture for Fog Computing," pp. 1–162, OpenFog Consortium, 2017.

[29] P. Pop, B. Zarrin, M. Barzegaran, S. Schulte, S. Punnekkat, J. Ruh, and W. Steiner, "The fora fog computing platform for industrial iot," *Information Systems*, vol. 98, p. 101727, 2021.

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

[31] "Microsoft edge zones," in *https://docs.microsoft.com/en-us/azure/networking/edge-zones-overview#edge-zones*. Accessed: February 2021.

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

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

[34] P. Barker and M. Hammoudeh, "A survey on low power network protocols for the internet of things and wireless sensor networks," in *International Conference on Future Networks and Distributed Systems (ICFNDS)*, pp. 1–8, 2017.

[35] V. Karagiannis, "Building a testbed for the internet of things," pp. 1–92, Alexander Technological Educational Institute of Thessaloniki, 2014.

[36] D. Charântola, A. C. Mestre, R. Zane, and L. F. Bittencourt, "Component-based scheduling for fog computing," in *International Conference on Utility and Cloud Computing Companion*, pp. 3–8, ACM, 2019.

[37] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource provisioning for iot application services in smart cities," in *2017 13th International Conference on Network and Service Management (CNSM)*, pp. 1–9, IEEE, 2017.

[38] V. Karagiannis and S. Schulte, "edgeRouting: Using compute nodes in proximity to route IoT data," *IEEE Access*, pp. 1–18, 2021.

[39] I. Farris, L. Militano, M. Nitti, L. Atzori, and A. Iera, "Mifaas: A mobile-iot-federation-as-a-service model for dynamic cooperation of iot cloud providers," *Future Generation Computer Systems*, vol. 70, pp. 126–137, 2017.

[40] N. Nikolakis, R. Senington, K. Sipsas, A. Syberfeldt, and S. Makris, "On a containerized approach for the dynamic planning and control of a cyber-physical production system," *Robotics and computer-integrated manufacturing*, vol. 64, p. 101919, 2020.

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

[42] Z. Á. Mann, A. Metzger, J. Prade, and R. Seidl, "Optimized application deployment in the fog," in *International Conference on Service-Oriented Computing (ICSOC)*, pp. 283–298, Springer, 2019.

[43] M. Z. Ahmed, A. H. AbdallahHashim, O. O. Khalifa, and M. J. Salami, "Border gateway protocol to provide failover in multihoming environment," *International Journal of Information Technology*, vol. 9, no. 1, pp. 33–39, 2017.

[44] P. Hu, M. Portmann, R. Robinson, and J. Indulska, "Context-aware routing in wireless mesh networks," in *International Conference on Context-awareness for Self-managing Systems (CASEMANS)*, pp. 16–23, ACM, 2008.

[45] V. Ricquebourg, D. Menga, D. Durand, B. Marhic, L. Delahoche, and C. Loge, "The smart home concept: our immediate future," in *International Conference on e-learning in Industrial Electronics (ICELIE)*, pp. 23–28, IEEE, 2006.

[46] N. Roy, A. Roy, and S. K. Das, "Context-aware resource management in multi-inhabitant smart homes a Nash h-learning based approach," in *International Conference on Pervasive Computing and Communications (PERCOM)*, pp. 1–11, IEEE, 2006.

[47] R. C. Shit, S. Sharma, D. Puthal, and A. Y. Zomaya, "Location of things (lot): A review and taxonomy of sensors localization in IoT infrastructure," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2028–2061, 2018.

[48] T. L. Duc, R. G. Leiva, P. Casari, and P.-O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–39, 2019.

[49] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, p. 107496, 2020.

[50] C. Guerrero, I. Lera, and C. Juiz, "A lightweight decentralized service placement policy for performance optimization in fog computing," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2435–2452, 2019.

[51] A. Aral and T. Ovatman, "A decentralized replica placement algorithm for edge computing," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 516–529, 2018.

[52] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*. Adaptive computation and machine learning, MIT Press, second ed., 2020.

[53] C. J. C. H. Watkins, *Learning from Delayed Rewards*. PhD thesis, King's College, University of Cambridge, UK, 1989.

[54] S. J. Majeed and M. Hutter, "On Q-learning convergence for non-markov decision processes," in *Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2546–2552, ijcai.org, 2018.

[55] E. Even-Dar and Y. Mansour, "Learning Rates for Q-learning," *Journal of Machine Learning Research*, vol. 5, pp. 1–25, 2003.

[56] V. Karagiannis, N. Desai, S. Schulte, and S. Punnekkat, "Addressing the node discovery problem in fog computing," in *Workshop on Fog Computing and the IoT (Fog-IoT)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[57] "Project repository," in *https://bitbucket.org/BasilKaragiannis/self-organizing-fog-computing*. Accessed online: February 2021.

[58] 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 *International Confer-*

ence on Automated Software Engineering (ASE), pp. 1114–1117, IEEE, 2019.

[59] “Refit dataset,” in *http://www.doi.org/10.17028/rd.lboro.2070091.v1*. Accessed: February 2021.

[60] M. Satyanarayanan, “How we created edge computing,” *Nature Electronics*, vol. 2, no. 1, pp. 42–42, 2019.

[61] A. Brown, M. Kolberg, and J. F. Buford, “Chameleon: an adaptable 2-tier variable hop overlay,” in *Consumer Communications and Networking Conference (CCNC)*, pp. 1–6, IEEE, 2009.

[62] Y. Li, F. Liu, Q. Chen, Y. Sheng, M. Zhao, and J. Wang, “MarVeLScaler: A Multi-View Learning based Auto-Scaling System for MapReduce,” *IEEE Transactions on Cloud Computing*, 2019.

[63] J. Kim, M. Son, and K. Lee, “MPEC: Distributed Matrix Multiplication Performance Modeling on a Scale-out Cloud Environment for Data Mining Jobs,” *IEEE Transactions on Cloud Computing*, 2019.

[64] D. Ding, X. Fan, Y. Zhao, K. Kang, Q. Yin, and J. Zeng, “Q-learning based dynamic task scheduling for energy-efficient cloud computing,” *Future Generation Computer Systems*, vol. 108, pp. 361–371, 2020.

Vasileios Karagiannis is with the Distributed Systems Group of TU Wien, Vienna, Austria. He has a BSc from the Alexander Technological Educational Institute of Thessaloniki, Greece, and a MSc from the University of Patras, Greece. Outcomes from his research on cloud computing, edge computing, and Internet of Things, have contributed to the publication of various scientific articles and patents.

Pantelis A. Frangoudis is a University Assistant with the Distributed Systems Group, TU Wien, Austria. He has been a researcher with the Communication Systems Department, EURECOM, France (2017-2019), and with team DIONYSOS at IRISA/INRIA Rennes, France (2012-2017), which he originally joined under an ERCIM “Alain Bensoussan” post-doctoral fellowship. He has a Ph.D. (2012) in Computer Science from AUEB, Greece. His interests include mobile and wireless networking, network softwarization, edge computing, and Internet multimedia.

Schahram Dustdar (Fellow, IEEE) is currently a Professor of computer science with the Distributed Systems Group, TU Wien, Vienna, Austria. From 2004 to 2010, he was an Honorary Professor of information systems with the University of Groningen, Groningen, The Netherlands, from 2016 to 2017, he was a Visiting Professor with the University of Sevilla, Seville, Spain, and in 2017, he was a Visiting Professor with the University of California at Berkeley, Berkeley, CA, USA. He is an elected member of the Academia Europaea, where he is Chairman of the Informatics Section. He was recipient of the ACM Distinguished Scientist Award (2009), the IBM Faculty Award (2012), and the IEEE TCSVC Outstanding Leadership Award (2018). He is the Co-Editor-in-Chief of the ACM Transaction on Internet of Things and the Editor-in-Chief of Computing (Springer). He is also an Associate Editor of the IEEE Transaction on Services Computing, the IEEE Transaction on Cloud Computing, the ACM Transaction on the Web, and the ACM Transaction on Internet Technology. He serves on the Editorial Board of IEEE Internet Computing and the IEEE Computer Magazine.

Stefan Schulte is Associate Professor and head of the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things at the Faculty of Informatics at TU Wien. His research interests span the areas of cloud computing and Internet of Things, and the application and extension of blockchain technologies. Findings from his research have been published in more than 100 refereed scholarly publications.