Future Generation Computer Systems 119 (2021) 7–19

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

JOI: Joint placement of IoT analytics operators and pub/sub message brokers in fog-centric IoT platforms

Daniel Happ [a,*], Suzan Bayhan [b], Vlado Handziski [a,c]

[a] Technische Universität Berlin, Germany
[b] University of Twente, The Netherlands
[c] R3 – Reliable Realtime Radio Communications GmbH, Germany

ARTICLE INFO

Article history:
Received 31 January 2020
Received in revised form 31 October 2020
Accepted 16 January 2021
Available online 22 January 2021

Keywords:
Publish/subscribe
Operator placement
IoT
Cloud
Fog
Brokerage

ABSTRACT

Internet of Things (IoT) systems are expected to generate a massive amount of data that needs to be processed. Given the large scale and geo-distributed nature of such systems, fog computing along with publish/subscribe (pub/sub) messaging has been proposed as possible solutions for coping with processing at scale. However, it is still unclear how practitioners can leverage the benefits of fog computing, e.g., how to optimally place data processing operators and pub/sub brokers. Moreover, current IoT systems typically rely on pub/sub brokers at the cloud, which might diminish the benefits offered by edge or fog processing as the communication between IoT operators has to be mediated by the brokers located in the cloud. To address this shortcoming, we propose to place the IoT application operators and the pub/sub brokers jointly on a network of nodes spanning from edge to the cloud considering various factors such as network topology or the locations of the IoT sensors and the consumers of the IoT applications. Different than the prior works, we specifically consider pub/sub brokers and their unique characteristics in the placement decision. First, we formulate the placement of operators and brokers jointly across edge, fog, and the cloud as a cost minimization problem. Next, we design two low-complexity heuristics. Our simulation results corroborate the argument that a placement in the cloud is usually a good option for IoT use cases, but also reveal the gap to the optimal solution in scenarios with heavier clustering of producers and consumers of sensor data. Studying the optimality gap shows that in such a setting heuristic solutions usually stay under a stretch factor of 2, with a worst case factor of 2.5 for a tabu-based solution and 2.85 for a greedy and a fixed placement in the cloud.

1. Introduction

In the emergent Internet of Things (IoT), billions of heterogeneous inter-connected devices will generate a massive amount of data that has to be distributed and processed efficiently to derive certain understanding from the sensor data, e.g., object recognition from the video cameras or finding the maximum temperature in a city during a time period. Other example applications for IoT analytics include forest fire detection [1], smart city video analytics [2], perimeter access control [3], or step counting in the health domain [4]. For the examples mentioned, the analytics applications can be partitioned into a graph of interdependent operators and then operators can be placed independently on the nodes available.

As the IoT vision is slowly becoming a reality, the number of deployments in domains such as smart buildings and smart cities is constantly rising. Despite significant spatial overlap or proximity, most of these deployments remain functionally and logically fully decoupled, leading to an emerging problem of wasteful overprovisioning of sensing, computation, and communication resources. This decoupling problem is further amplified by the currently dominant cloud-centered architectural pattern that promotes upstreaming of raw sensing data from the device layer directly to the cloud, leaving the cloud as the only "data peering" point for applications that would like to combine data.

However, cloud-based IoT suffers from several shortcomings. First, many applications such as virtual reality are sensitive to the latency introduced by relaying data and processing over the cloud [5]. Additionally, the upstream bandwidth required for sending raw data might prohibit this approach where no broadband connection is available and use of alternative technologies such as cellular networks is not desirable due to cost or performance reasons. Other shortcomings of the cloud-based approach include unknown or ambiguous privacy policies. The emerging fog-centered architectural pattern [6] envisions a data processing

* Corresponding author.
E-mail addresses: daniel.happ@tu-berlin.de (D. Happ), s.bayhan@utwente.nl (S. Bayhan), vlado.handziski@r3coms.com (V. Handziski).

https://doi.org/10.1016/j.future.2021.01.026
0167-739X/

pipeline that processes IoT at least partially before reaching the cloud, e.g., within the core network of an ISP. Conceptually, this opens an opportunity for (additional) data peering closer to the sources of data, thus, resulting in a more efficient sharing of the available hardware resources, reduced latency, and increased reliability and privacy. To maximize the potential, future IoT applications will have much finer granularity and will be deployed across edge, fog, and cloud.

Despite the availability of a large body of work on the placement of operators across edge, fog, and cloud, these studies overlook the communication model between the operators. We believe that the decoupling benefits provided by publish/subscribe (pub/sub) overlays can serve a key role in streamlining the flow of data between the application components or between components of different applications [7,8]. An important aspect in the design of such overlays is the number and placement of the core broker servers that facilitate matchmaking between the producers (publishers) and the consumers (subscribers) of the relevant data streams.

Although a plethora of works investigate the problem of operator placement, placing operators and brokers optimally in the pub/sub architecture is not straightforward and, to the best of our knowledge, is not jointly considered in the literature. The prior research on optimal operator placement, e.g., [6], has several shortcomings. First, most of these studies assume either a fixed broker placement or assume direct communication between an IoT data generator (referred to as *producer* or *publisher*) and the *consumer* (referred to as *subscriber*) of the generated data. The latter might result in scalability and energy inefficiency problems especially for resource-constrained IoT equipment [9]. As for brokers, their location has implications on the optimal placement of the operators and might impose some limitations if the broker placement is not optimized. In contrast to previous work, we show that a joint optimization of the placement of the application components (operators) and the pub/sub brokers across the edge, fog and cloud can lead to significant benefits compared to isolated treatment of the problems of application and middleware decomposition and deployment. We also highlight that our approach is specifically tailored to the pub/sub use case by showing its expressiveness in comparison to the previous work, i.e., there are parameters and constraints that cannot be easily achieved by the existing approaches.

Since we anticipate future IoT systems to be characterized by considerable heterogeneity, we do not expect a universal platform for operator definition and placement or a universal global pub/sub system. We instead consider interconnected silo-solutions that have fixed points where messages can be exchanged between different organizations. We therefore consider the placement problem from the viewpoint of the owner of one of those silos, which can place its brokers and operators freely on a set of hardware gateways as well as on various multi-tenancy cloud and fog providers with limitations on the underlying network, especially the available throughput between different nodes.

Our contributions in this paper are as follows:

- For a service provider, we formulate the problem of joint deployment of brokers and analytics operators on edge, fog, and cloud resources as an optimization problem. Different than prior works, e.g., [10], we do not assume a direct communication between two operators or a given broker deployment [11]. We find the optimal placement for pub/sub brokers.
- Given the complexity of the placement problem, we develop two heuristic solutions to the placement problem and quantify their optimality gap under various scenarios.
- We provide a thorough evaluation via simulations using realistic data for analytics applications and network topologies. Our analysis suggests that with increasing spatial locality of the IoT data providers (i.e., sensors and their consumers), the traditional cloud-based analytics become less desirable due to the resulting higher delay. In other words, placing the brokers and the IoT analytics operators jointly as proposed in this paper offers more benefits for such cases while a conventional cloud-based approach can fit better to cases where large spatial distribution of publishers and subscribers are expected.
- Our analysis on a smart crosswalk scenario corroborates our argument that the optimal broker locations might be different than the optimal location for operators, thereby motivating the need for joint placement.

The rest of the paper is structured as follows. Section 2 introduces the considered system model along with the key assumptions. Section 3 presents the joint placement problem as a linear integer programming problem. In Section 4, we introduce two low-complexity heuristics to the joint placement problem. Section 5 evaluates the run time, the optimality gap in various scenarios and the distribution of deployments between edge, fog and cloud obtained by the presented solutions. Section 6 summarizes the related work while Section 7 concludes our work.

2. System model

Consider an application owner which offers multiple IoT applications, each consisting of a set of operators. The considered system model consists of sensor data producers, applications, and data consumers. Physical sensor devices act as producers by measuring real world phenomena. Analytics applications process this raw data to generate meaningful output that is valuable to interested service endpoints, end-users, or actuators as data consumers. In contrast to prior works, we envision and specifically consider that IoT deployments will resemble a sharing economy, so the output of one sensor or one sub-result of an application will be used by multiple applications and their users. As a consequence, we assume that applications internally use the pub/sub paradigm for all messaging needs. Our solution, referred to as JOI, takes into account one or multiple application graphs and the underlying network topology consisting of nodes at the edge, fog, and the cloud. As Fig. 1 shows, it then outputs a placement of the operators and pub/sub brokers on the nodes. In the following, we define the key components of our system in more detail: nodes, operators, and brokers.

Nodes: We consider a set of devices denoted by $N = \{\cdots, n_k, \ldots\}$ where n_k is node k and can be a cloud-based virtual machine at the *cloud layer*, fog virtual machine operated by Internet service providers (ISP) at the *fog layer*, and on-premise sensor gateway at the *edge layer*. We assume that the application owner can deploy operators and brokers on those nodes in an on-demand pay-as-you-go fashion. Nodes might differ from each other in various ways, e.g., available communication bandwidth or network latency to other nodes. We expect that the application owner knows the following properties of each node k: (1) the available maximum capacity $R_{k,q}$ of a specific resource $q \in Q$ (e.g. CPU or memory); (2) the available uplink ($\beta_k^{\uparrow}$) and downlink ($\beta_k^{\downarrow}$) capacity of the node to the access network; (3) a cost factor f_k indicating a cost ratio with regard to a reference machine representing the cost of deploying an IoT *operator* or a broker at this node.

Operators: Let us define an operator as the smallest deployable software component [10] which corresponds very broadly to an arbitrary computation on a set of input streams that create

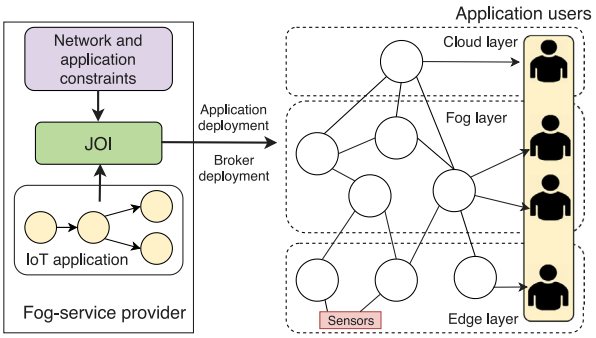


Fig. 1. Overview of JOI. Taking the IoT application graph, network and application constraints, JOI decides where to deploy operators and brokers.

one or more output streams. Then, we abstract publishing sensor devices, processing tasks, subscribing applications or services of all applications under the same application owner as generic *operators* o_i. Moreover, we define an application as a graph of operators. To represent the relationship between the operators of an application, we define a binary dependency matrix $D_{i,j}$ as follows: $D_{i,j} = 1$ if o_i depends on o_j and zero otherwise. Since purely publishing operators, i.e., IoT sensors, have no input data stream, they do not depend on other operators, i.e., $D_{i,*} = 0$ for such o_i. Similarly, IoT subscribers have no other operators depending on them and no output, i.e., $D_{*,i} = 0$ for such o_i. Each operator $o_i \in O$ is characterized by an associated cost c_i for calculating one output message, an average output rate r_i (per second), and the resource demand $\Gamma_{i,q}$ for several resources $q \in Q$. Resources usually include CPU and memory requirements. They could also include further requirements such as a certain type of hardware that is required for the particular operator, e.g., a dedicated graphic processor for machine learning or video encoding in hardware. Note that an application owner can profile its applications first to observe the dynamics on the needed resources, e.g., maximum and average memory usage. Then, it can set the resource demands accordingly. A conservative approach would be to set the resource demand taking the maximum resource usage of an operator, which might result in significant overprovisioning. Alternatively, the application owner can set the requirement according to the average resource usage with some overprovisioning, e.g., 20% above the average resource usage.

Brokers: Because of the benefits offered by the decoupling between producers, operators and consumers, we assume a pub/sub-based IoT system with matchmaking brokers. We assume that each operator publishes to exactly one broker, from where the data is disseminated to one or multiple subscribers.

3. JOI: Joint placement of IoT brokers and operators

The problem we aim to solve is the joint placement of brokers and operators on a given set of nodes such that the cost of operating the network with the given set of analytics applications, subscribers, and publishers is minimized. To this end, we have to decide on a mapping of operators to nodes as well as placement of message brokers connecting the operators. The application owner needs the following two network parameters: (i) matrix $C_{k,k'}$ denoting the cost (e.g., latency) of sending one byte from n_k to $n_{k'}$; and (ii) the matrix $\beta_{k,k'}$ denoting the available bandwidth between nodes n_k and $n_{k'}$. The application owner can perform active or passive measurements to acquire such information. Moreover, due to the network dynamism, measurements should be performed periodically to capture the short term characteristics as well as the long-term network state.

Now, let us introduce our binary decision variables:

Table 1
Variables and parameters.

Symbol	Definition
Parameters	
$N = \{\cdots, n_k, \ldots\}$	Set of nodes in the network
$O = \{\cdots, o_i, \ldots\}$	Set of operators to be deployed
$D_{i,j}$	1 iff o_i depends on output of o_j
δ_i	Latency requirements of o_i
c_i	Cost to generate one output message of o_i
r_i	Output data rate of o_i
Q	Set of constrained resources (CPU, memory)
$\Gamma_{i,q}$	Demands of operator o_i for resource $q \in Q$
Θ_q	Demands for resource $q \in Q$ when brokering
$R_{k,q}$	Available capacity of resource $q \in Q$ on n_k
$\beta_k^{\uparrow}$, $\beta_k^{\downarrow}$	Available uplink and downlink bandwidth on n_k
f_k	Cost factor of n_k
$C_{k,k'}$	Cost of sending one byte from n_k to $n_{k'}$
$\beta_{k,k'}$	Available bandwidth between n_k and $n_{k'}$
$L_{i,k}$	1 iff the location of o_i is fixed on n_k
B	Maximum number of brokers deployed
$\mathcal{B}_k$	1 iff there should be a broker fixed on n_k
Decision Variables	
$x_{i,k}$	1 iff o_i is on n_k
$y_{i,k}$	1 iff o_i publishes to a broker on n_k
Helper Variables	
$z_{i,k,i',k'}$	1 iff o_i is on n_k and $o_{i'}$ publishes to a broker on $n_{k'}$
b_k	1 iff there is a broker on n_k
ϕ_i	1 iff the o_i generates output (i.e. is no subscriber)
d_i	Latency of executing o_i including all depending ops

- $x_{i,k}$ showing whether o_i will be placed on n_k,
- $y_{i,k}$ is the decision variable showing if o_i publishes to a broker on n_k, which implicitly means that we deploy a broker at n_k.

Additionally, we will introduce an auxiliary variable $z_{i,k,i',k'}$ which is defined as follows: $z_{i,k,i',k'} = x_{i,k} \times y_{i',k'}$. Hence, it is 1 if o_i is on n_k and $o_{i'}$ publishes to a broker on $n_{k'}$. Let b_k denote whether n_k hosts a broker, i.e., $b_k = 1$ if n_k hosts a broker and 0 otherwise.

A feasible placement obviously has to consider the capacity constraint of both the network and the nodes themselves. Additionally, to represent sensor devices with a fixed location and cloud-based value-added services, we introduce $L_{i,k}$ which enforces a fixed location on node n_k for some operator o_i. To represent operators with an output, we introduce a binary auxiliary variable ϕ_i, which is 1 if the operator has an output. We allow to limit the number of brokers using parameter B and also allow to fix the broker locations using parameter $\mathcal{B}_k$. Please refer to Table 1 for a list of the key variables.

We now outline our optimization problem and start by presenting the constraints followed by our objective function. We define our variables as follows:

$$x_{i,k} \in \{0, 1\}\ ,\forall k \forall i, \tag{1}$$

$$y_{i,k} \in \{0, 1\}\ ,\forall k \forall i, \tag{2}$$

$$b_k \in \{0, 1\}\ ,\forall k, \tag{3}$$

$$z_{i,k,i',k'} \in \{0, 1\}\ ,\forall i \forall k \forall i' \forall k'. \tag{4}$$

Next, to enforce $x_{i,k} = 1$ for those operators that have fixed locations (the subscribers and publishers) with $L_{i,k} = 1$, we define the following constraint:

$$x_{i,k} \geq L_{i,k}\ ,\ \forall o_i \in O, \forall n_k \in N. \tag{5}$$

Since we aim at deploying each operator at exactly one node, we state this constraint as follows:

$$\sum_{n_k \in N} x_{i,k} = 1\ ,\ \forall o_i \in O. \tag{6}$$

To cover also the cases where the brokers are already deployed, we introduce the following constraint:

$$b_k \geq \mathcal{B}_k \ , \forall n_k \in N, \tag{7}$$

which enforces $y_{i,k} = 1$ for at least one of the operators if $\mathcal{B}_k = 1$, i.e., sets a fixed broker. With (7), our problem will find the optimal placement of the IoT analytics given a fixed broker placement.

Moreover, with the following constraints, we ensure that only operators with some output publish at a broker:

$$\sum_{n_k \in N} y_{i,k} = \phi_i \ , \forall o_i \in O \tag{8}$$

$$\phi_i \leq \sum_{o_j \in O} D_{j,i} \ , \forall o_i \in O \tag{9}$$

$$|O| \cdot \phi_i \geq \sum_{o_j \in O} D_{j,i} \ , \forall o_i \in O. \tag{10}$$

Recall that ϕ_i yields 1 only if the operator has an output, i.e., another operator j depends on o_i.

As there could be a limitation on the number of brokers, we introduce the following constraints:

$$\sum_{n_k \in N} b_k \leq B \tag{11}$$

$$b_k \leq \sum_{o_i \in O} y_{i,k} \ , \forall n_k \in N \tag{12}$$

$$|O| \cdot b_k \geq \sum_{o_i \in O} y_{i,k} \ , \forall n_k \in N \tag{13}$$

where Consts. (12) and (13) together ensure that b_k equals 1 if there is some operator publishing at n_k.

Finally, we define $z_{i,k,i',k'} = x_{i,k} \times y_{i',k'}$ using linear functions as follows:

$$z_{i,k,i',k'} \leq x_{i,k}, \tag{14}$$

$$z_{i,k,i',k'} \leq y_{i',k'}, \tag{15}$$

$$z_{i,k,i',k'} \geq x_{i,k} + y_{i',k'} - 1. \tag{16}$$

As each node has limited resources, Const. (17) asserts that the total resources consumed by both the operators and the broker deployed on n_k are lower than the capacity of this node per resource type:

$$\sum_{o_i \in O} \left(x_{i,k} \cdot \Gamma_{i,q} + y_{i,k} \cdot \Theta_q \cdot r_i \right) \leqslant R_{k,q} \ , \forall q \in Q, \forall n_k \in N, \tag{17}$$

where the first term shows the resource consumption of an operator and the second component shows the brokering cost which depends on the volume of output data generated by the operators publishing to this broker.

Next, Const. (18) specifies that the volume of data transmitted to each node must be lower than the corresponding node's downlink capacity:

$$\overbrace{\sum_{o_i \in O} \left(\sum_{o_j \in O} D_{i,j} \cdot r_j \cdot (x_{i,k} - z_{i,k,j,k}) \right)}^{\text{Input Data}} + \underbrace{r_i \cdot (y_{i,k} - z_{i,k,i,k})}_{\text{Brokerage}} \leq \beta_k^{\downarrow} \quad \forall n_k \in N \tag{18}$$

where the first term in (18) corresponds to the total amount of input data to a particular node for all the operators it hosts. Since the operators colocated with their brokers on the same node can send their data locally, first term of (18) excludes such operators, i.e. (1-$y_{i,k}$). The second term of (18) represents the input traffic from operators to the broker hosted on this node. Note the term $(1 - x_{i,k})$ excludes the operators who are located on the same node. This constraint is crucial to avoid network congestion especially at the fog and edge layers where the network devices might be limited in their bandwidth resources.

Similarly, we formally state that the aggregate outgoing traffic from a node must be lower than the uplink capacity of this node as follows:

$$\overbrace{\sum_{o_i \in O} r_i \cdot (x_{i,k} - z_{i,k,i,k})}^{\text{Operator to Broker}} + \underbrace{\sum_{o_j \in O} D_{j,i} \cdot r_i \cdot (y_{i,k} - z_{j,k,i,k})}_{\text{Broker to Operators}} \leq \beta_k^{\uparrow} \ , \forall n_k \in N. \tag{19}$$

Finally, to ensure that the total traffic from one node to the other does not exceed the capacity of the path connecting them, we introduce the following constraint:

$$\sum_{o_i \in O} z_{i,k,i,k'} \cdot r_i + \underbrace{\sum_{o_{i'} \in O} z_{i',k',i,k} \cdot D_{i',i} \cdot r_i}_{\text{Broker at } k, \text{ next Operator at } k'} \leq \beta_{k,k'}$$

$$\forall n_k \in N, \forall n_{k'} \in N \setminus n_k. \tag{20}$$

Please note that two nodes n_k and $n_{k'}$ might be connected via multiple hops. However, the application owner usually does not know the exact network topology. Instead, it can probe the link capacity between a pair of nodes. Hence, rather than considering the total routed traffic over each link, we consider the end-to-end capacity between two nodes in Const. (20). Note that we can assert additional constraints for each operator's location by setting $y_{i,k}$ values explicitly. For example, if the operators have some privacy requirements such that a public cloud provider violates this privacy requirement, we can set the corresponding $y_{i,k}$ to zero to avoid our algorithm placing the operator to the cloud.

Next, we define the delay of a subscribing operator d_i recursively as follows:

$$d_i = \max_{D_{i,j}=1} \left(d_j + \left(\sum_{n_k \in N} \sum_{n_{k'} \in N} z_{j,k,j,k'} C_{k,k'} + z_{i,k,j,k'} C_{k',k} \right) + \sum_{n_k \in N} r_i x_{i,k} f_k c_i \right). \tag{21}$$

In (21), d_i represents the time needed to complete this operator's task and depends on three components: (i) time to receive the input data from all other operators that this operator o_i depends on, (ii) the transmission time of the published data from all operators that o_i depends on, (iii) and finally the time to execute this operator itself on the hosting node. Since an operator cannot start executing its task before it receives all input from the operators it depends on, we have to consider the maximum latency between o_i and each o_j where $D_{i,j=1}$. Hence, the first term in (21) takes the maximum latency.

Since applications may differ in terms of their tolerance to delay, we introduce δ_i to specify the maximum latency requirement of an application that users can tolerate. Note that an application provider can also express different requirements for different users or operators in between publishers and subscribers. We express this constraint as follows:

$$d_i \leq \ \delta_i \ \forall i. \tag{22}$$

By considering the delay requirement of each application, we ensure that the resulting placement satisfies the quality-of-service (QoS) requirements of the provided applications. Hence, in the objective function, we opt to minimize the sum of end-to-end delays of all subscribers.[1] We formally state the objective function as the sum of delays as follows:

$$min \sum_{o_i \in O} (1 - \phi_i) d_i. \tag{23}$$

For completeness, let us present the joint placement of operators and brokers as follows:

$$\min_{x,y,b,z} \sum_{o_i \in O} (1 - \phi_i) d_i, \tag{24}$$

$$s.t. \quad (1) - (22). \tag{25}$$

3.1. Discussion on expressiveness

As stressed earlier, we are not aware of other works that consider the joint placement of operators and brokers. However, it might be possible to solve this problem by treating brokers as generic operators. In this section, we discuss why this is not sufficient.

First, our model does not only solve the placement of brokers, but also encodes the operator to broker association in the variable y. When trying to solve the joint placement problem by treating brokers as operators, this association has to be provided in advance in the form of a suitable dependency matrix. Deciding in advance before having any insights about the possible placement of the operators which broker they might share with other operators would be a great challenge on its own.

A solution might be not to share brokers at all, but that would make modeling synergy effects from sharing a single broker challenging. Also, imposing limits on the total number of brokers would be difficult, but is easily possible in our model. Therefore, the model presented here, as well as the heuristics that follow, are more expressive than other works, even if brokers are treated as generic operators.

3.2. Computational complexity

Inspired by the proof in [12], in the following, we formally show that the problem in (24) and (25) is NP-hard by reduction from the Subgraph Isomorphism Problem (SIP). We start by recapping a concise simplified definition of the joint operator and broker placement problem as a decision problem, proceed to define SIP and finally show how SIP can be reduced to our problem.

Definition 1. Assume a set of nodes $N = \{n_k\}$. Each node k has available resources $R_{k,q}$ $\forall q \in Q$ associated with it, as well as a maximum access bandwidth in both directions $(\beta^{\uparrow}, \beta^{\downarrow})$. The network between the nodes is defined by two matrices for cost of data transmission $(C_{k,k'})$ and available bandwidth $(\beta_{k,k'})$ between node k and k'. Applications are defined by operators $O = \{o_i\}$ with a dependency matrix $D_{i,j}$, output rate r_i and resource demands $\Gamma_{i,q}$. Given these inputs, a solution of the problem is defined as an assignment of operators and brokers to nodes that adhere to the resource and bandwidth constraints.

[1] An application provider can opt for other objective functions depending on its primary goals, e.g., the (monetary) cost of deploying operators and brokers and additional charges for data traffic.

Definition 2. Let $H = (V_H, E_H)$ and $G = (V, E)$ be graphs. A subgraph isomorphism from H to G is a function $f : V_H \rightarrow V$ such that if $(u, v) \in E_H$, then $(f(u), f(v)) \in E$. The Subgraph Isomorphism Problem (SIP) is, given H and G, to determine whether there is a subgraph isomorphism from H to G. Subgraph Isomorphism is an NP-complete problem.

Theorem 1. *The joint problem of broker and operator placement is NP-complete.*

Proof. The general idea of the proof is that JOI solves SIP by finding a deployment of the application vertices $o \in O$ with edges $D_{i,j}$ on the infrastructure given by the nodes $n \in N$ and their edges $\beta_{k,k'}$. Assume a SIP instance is given. This SIP can be reduced to our joint placement problem in polynomial time as follows:

1. We first interpret the vertices given by $v \in V$ of the graph G as nodes $n \in N$ in our problem domain. The given edges E correspond to the connectivity of the nodes in JOI's domain. Hence, we change all edges $(u, v) \in E$ into links with latency $C_{k,k'} = C_{k',k} = 0$ and bandwidth $\beta_{k,k'} = \beta_{k',k} = 1$. The bandwidth between nodes where $(u, v) \notin E$ is set to $\beta_{k,k'} = \beta_{k',k} = 0$ with 0 cost as well.
2. We interpret all vertices in V_H as operators with all resource demands set to 1 and output rate 1. The dependency matrix $D_{i,j}$ is given by the edges E_H: We iterate over all vertices in V_H and add a dependency for each edge in E_H involving these vertices if this edge has not been added before. This way, the resulting matrix $D_{i,j}$ is an acyclic directed graph corresponding to the original undirected graph H.
3. Since there are no additional attributes in the SIP instance, we set all available resources $R_{k,q} = 1, \forall n_k \in N \; \forall q \in Q$ in our JOI problem to make sure that exactly one operator can be deployed on each node. We further set all broker costs $(\in \Theta)$ to 0 and the auxiliary parameter $f = 1$ for all nodes.

If one or multiple solutions exist, one of them will be the output of the JOI problem. Brokers will be placed either with their corresponding operator or a single downstream operator and can be safely ignored. The placement of the operators and their dependency on a network of nodes and interconnections then can be easily translated back to the original SIP problem. Hence, iff a solution for the JOI problem exists, a corresponding solution for the SIP problem exists. Therefore, a solution for the JOI problem also solves the SIP problem. It follows that our joint problem of broker and operator placement is also NP-complete. □

Indeed, solving the placement problem with state-of-the-art commercial linear integer problem solvers becomes impractical for larger topologies (e.g., about 50 nodes and 50 operators on today's desktop hardware). Hence, we propose two heuristics in the following.

4. Heuristic approaches

Similar to the optimal solution, our heuristics are centralized and use global knowledge of the system to make placement decisions. Our baseline is the traditional cloud-based operation, referred to as CLOUD, wherein brokers and operators reside in the cloud. We propose a greedy heuristic (GREEDY) and a tabu search (TABU) that uses the greedy solution as its input.

4.1. Greedy approach (GREEDY)

The main objective of GREEDY is to locally minimize the delay between each operator and operators depending on it. Algorithm

Algorithm 1 JOI Greedy Placement

```
1: function Placement(x, y, app, topology)
2:     opstoplace ← operators without node
3:     brokerstoplace ← operators without broker
4:     if opstoplace = ∅ ∧ brokerstoplace = ∅ then
5:         working ← CheckAssignment(x, y, app, topology)
6:         return working, x, y
7:     if |opstoplace| < |brokerstoplace| then
8:         working, y ← BrokerPlacement(x, y, app, topology)
9:     else
10:        working, x ← OperatorPlacement(x, y, app, topology)
11:    return working, x, y
```

Algorithm 2 JOI Greedy Operator Placement

```
1: function OperatorPlacement(x, y, app, topology)
2:     toplace ← operators without node
3:     if toplace = ∅ then
4:         working ← CheckAssignment(x, y, app, topology)
5:         return working, x
6:     toplace ← SortOperators(toplace, criteria_op)
7:     op ← toplace_0
8:     nodestotry ← SortNodes(n ∈ N, criteria_node)
9:     for all node ∈ nodestotry do
10:        x_op ← node
11:        working ← CheckAssignment(x, y, app, topology)
12:        if ¬working then
13:            continue
14:        working, x, y ← Placement(x, y, app, topology)
15:    return false, x
```

Algorithm 3 JOI Greedy Broker Placement

```
1: function BrokerPlacement(x, y, app, topology)
2:     toplace ← operators without broker
3:     if toplace = ∅ then
4:         working ← CheckAssignment(x, y, app, topology)
5:         return working, y
6:     toplace ← SortOperators(toplace, criteria_op)
7:     op ← toplace_0
8:     nodestotry ← SortNodes(n ∈ N, criteria_node)
9:     for all node ∈ nodestotry do
10:        y_op ← node
11:        working ← CheckAssignment(x, y, app, topology)
12:        if ¬working then
13:            continue
14:        working, x, y ← Placement(x, y, app, topology)
15:    return false, y
```

1 shows the steps of GREEDY with the functions of placing operators and brokers in Algorithm 2 and Algorithm 3, respectively. The order of operators and brokers to place and the criteria for the placement are kept generic and can be adapted for the considered use case.

GREEDY first places fixed operators and fixed brokers to their corresponding nodes. It then assigns a stratum number to each operator where the stratum represents the distance of the operator in hops from a data source. GREEDY sorts operators by their stratum number and places operators with the highest stratum first on the closest feasible node to their sink(s), i.e., subscriber(s). In this process, GREEDY ignores operators that are not yet placed and does not consider the position of brokers initially. It searches the solution space in a depth-first fashion: If an operator cannot be placed on any node given the current assignment of already-placed operators, the operator that was placed last is moved to the next best node. This step is repeated recursively until either a valid solution is found or every possible combination has been tried, which would mean that there is no feasible assignment.

After placing an operator, the next objective is to place the corresponding broker efficiently. GREEDY places the broker considering the already-placed operators and ignoring those that are yet to be placed. After assigning an output broker for each operator, GREEDY checks if the constraint on the number of brokers is violated. In case of violation, GREEDY merges two brokers that are closest to each other in terms of latency until the constraint on the total number of brokers is satisfied. In the last step, the algorithm moves brokers to their optimal position in the topology based on the position of the operators they serve.

4.2. Tabu-like search (TABU)

Next, we adapt a tabu search approach to our problem [13]. TABU starts with a known feasible solution as its input and aims at improving this solution iteratively by searching for a better solution in the neighborhood of the current solution. In our case, it starts once with GREEDY and once with CLOUD and takes the best of these two solutions. In general, tabu search is an iterative approach, where in each round a neighborhood of possible solution is considered. The neighborhood of solutions in our approach is determined as follows. In each round, TABU might take one of the following three actions to improve the current solution: (1) change operator position, (2) change operator/broker assignment, and (3) change broker position. We choose the actions one after the other.

For choice (1) and (2), TABU chooses an operator randomly. For choice (1), the neighborhood is given by every possible movement of the operator. For (2), the neighborhood is every possible reassignment to an existing broker. Similarly, in case of (3), TABU chooses a broker randomly and considers every possible movement of the broker to find an improved solution. The search uses a short term memory that stores the last u solutions as a tabu list. TABU in general chooses the best solution of the neighborhood as the next candidate, except if the solution was already tried recently, which is a tabu. In that case, the next best solution that was not tried recently is chosen, which makes the algorithm consider sub-optimal solutions to avoid getting trapped in a local optimum. The candidate is chosen as the starting point for further iterations and its utility is compared to the current overall best solution, which it replaces if it is better. TABU stops after a fixed number v consecutive iterations without improvements.

5. Performance evaluation

We evaluate the performance of the proposed approaches via system-level simulations on our custom-made Python simulator. Our goal in assessing the performance of our system is to gain further insights on the following two questions: (1) How much is the gain of joint optimization of broker and operator locations compared to more traditional approaches (e.g., cloud based)?; and (2) Which heuristics provide the best performance for solving the joint problem?

To solve the problem in (23) optimally, we use Gurobi version 8.0 [14] via its C++ interface. We perform simulations on an Intel Core i7-4790 CPU @ 3.6 GHz with 4 cores and 8 threads with 15.6 GiB of RAM running an Ubuntu 16.04 LTS (Xenial Xerus) 64-bit operating system. We run non-deterministic heuristics 20 times for statistical significance.

5.1. Network topology

To address the above-listed research questions, we first need to specify the parameters used for the generation of the network topology and applications. We assume that the system in terms of latency spans a country or region of a similar size as the continental United States. As Fig. 2 shows, edge devices are connected to only one ISP while ISPs (in the fog layer) are interconnected and have a high-capacity connection to the cloud.

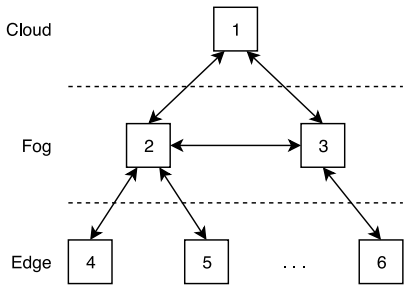


Fig. 2. Three layered network topology with interconnected nodes in the cloud, fog and edge layer.

Table 2
Bandwidth and round-trip-time between nodes.

	Bandwidth		Round-trip-time	
	Minimum	Maximum	Minimum	Maximum
Edge-Fog	1 MBit/s	100 MBit/s	10 ms	30 ms
Fog–Fog	100 MBit/s	1 GBit/s	5 ms	70 ms
Fog–Cloud	100 MBit/s	10 GBit/s	5 ms	35 ms

Table 3
Resources on different nodes.

	CPU Score		Memory	
	Minimum	Maximum	Minimum	Maximum
Edge	420	3800	1 GB	4 GB
Fog	5500	12 000	8 GB	32 GB

To model the bandwidth and latency values among nodes realistically, we consult available public sources and use the values listed in Table 2. The values are derived for a country like the United States or a similar region of the same size. For the average Edge to Fog bandwidth, we use the Steam global traffic statistics,[2] which shows an average downlink bandwidth of about 50 MBit/s (46.7 MBit/s) and set a minimum and maximum bandwidth to be 1 MBit/s and 100 MBit/s respectively. Since residential connections are usually asymmetric, we set the uplink bandwidth to one fourth of the downlink bandwidth. For the bandwidth in the Fog and between Fog and Cloud, we use common values for 100BASE-TX, 1000BASE-T and 10GBASE-T Ethernet.

For the estimation of the delay, we use a public looking glass server from Hurricane Electric.[3] The round-trip-time (RTT) between the west (Freemont, CA) and east (New York, NY) coast of the continental US was estimated to be around 70 ms between different ISPs. The RTT between the Hurricane Electric Freemont, CA location and the Amazon Web Services data center in Virginia, however, is only about 35 ms.

To model the CPU and memory resources of each node type listed in Table 3, we take the Passmark score[4] of common CPU models and memory configurations. Edge devices are considered to be similar to current low-power firewall devices and fog nodes to be of workstation/server type. We assume that an edge node has a computing power between an ARM Quadcore (score 420) and an Intel Atom C2750 (score 3800). An Intel Atom E3845 lies between the two with a score of about 1500. For memory, we assume between 1 and 4GB of RAM. For the fog, we take common server hardware as an estimate and assume that nodes have a CPU score between the low-end Intel Xeon E5-1603 (score 5500) and the high-end Xeon W2125 (score 12000) with a memory configuration between 8GB and 32GB.

[2] https://store.steampowered.com/stats/content/
[3] https://lg.he.net/
[4] https://www.passmark.com/

5.2. Applications

As we aim at obtaining as general results as possible, we prefer considering a broad spectrum of synthetic applications that follow a common structure rather than a very specific application graph as appears in the literature, e.g., [1–4]. We consider the following three cases depicted in Fig. 3 as the application graphs:

- A **fanout topology** where a set of processing operators needs the input of every publisher in the system and gives its output to every subscriber;
- A **sequence topology** where the processing operators form a chain of operators that must run as a sequence;
- A **random topology** which in our case is not allowed to have any cycles.

We choose the CPU requirements between 100 and 2000 in terms of CPU score. Likewise, each operator uses between 0.05 and 1GB of RAM and has an output rate between 0.001 and 1MBit/s. Publishers and subscribers are fixed on the edge.

5.3. Scenarios

To address the research questions we raised earlier, we will compare the following schemes:

- OPT: the optimal joint solution we derived by solving the formulated problem using Gurobi solver,
- CLOUD: cloud-only deployment where all brokers and analytics operators are hosted in the cloud. This represents the current conventional cloud-based IoT systems. We will use it as our baseline.
- Our heuristics GREEDY and TABU.

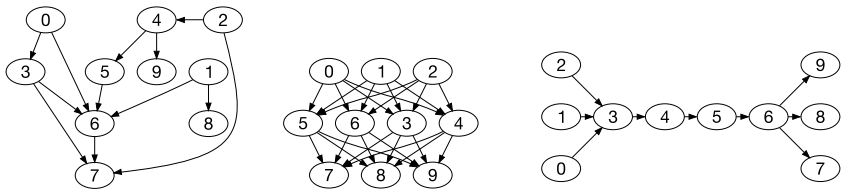
Through evaluating these schemes, we will be able to answer if the joint optimization has a significant advantage over traditional approaches. The comparison of OPT with heuristics shows the trade-off between complexity and quality of solution.

Before analyzing the quality of the solutions yielded by each approach, let us first analyze the CPU time each approach needs to find a feasible placement. To observe the scalability of each approach, we will increase the network size from 5 to 50 nodes, while keeping the number of operators fixed at 10. We use first random application topology and report the CPU and memory usage of each approach. Next, we analyze the impact of the type of the application graph (fanout, sequence, random). We further investigate the impact of publisher/subscriber clustering. We consider applications with 40 operators including 5 publishers and 5 subscribers. To account for different spatial locality of services, we model three scenarios: *full/heavy clustering* where publishers and subscribers are connected to the same node; *medium clustering* where publishers and subscribers are on one node and its next two neighbors; and *no clustering* where publishers and subscribers are distributed across all nodes. A more detailed analysis of the impact of clustering is presented in a related paper [15].

As performance indicators, we report the following metrics: optimality gap denoting the ratio of the utility value achieved by a heuristic over the utility of OPT, utilization of each network tier by operators and brokers, and delay distribution between operators and their consumers.

5.4. Results & discussion

Fig. 4 shows the CPU time of each approach with increasing network size. We omit the CLOUD in this figure as it places all operators in the cloud. We observe that CPU time of OPT and TABU tend to rise exponentially with increasing network size while GREEDY needs significantly less time for finding a



(a) Random (b) Fanout (c) Sequence

Fig. 3. Considered application dependency graphs; operators 0–2 with no incoming links are publishers, operators 7–9 with no outgoing links are subscribers.

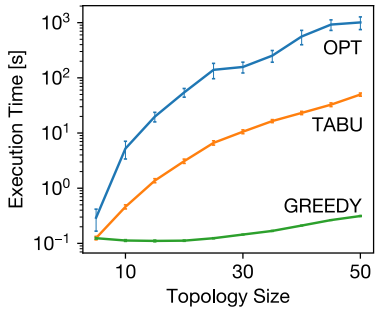


Fig. 4. Analysis of the CPU time on a reference machine to solve JOI for a particular topology size.

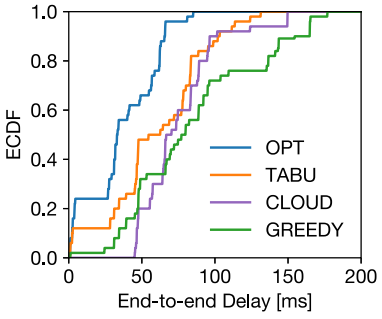
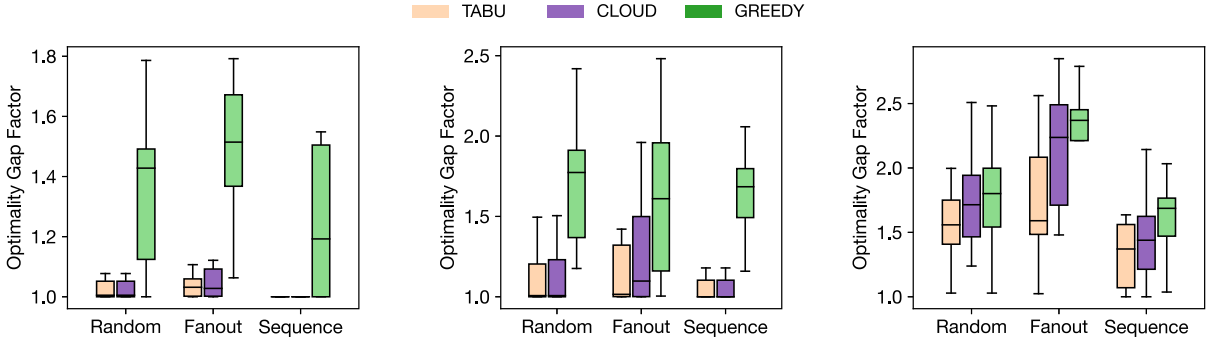


Fig. 5. An example delay distribution of the solutions.

placement. For the largest topology considered, OPT takes about 10^3 seconds; more than an order of magnitude longer time to return its solution as compared to TABU with about a minute. While TABU has a higher running time compared to GREEDY (around two orders of magnitude difference), we later show that it outperforms the latter significantly and therefore could provide a trade-off between scalability and performance. Moreover, please note that our problem is designed to be solved offline. Therefore, we expect that a fog-service provider has sufficient computation resources to execute TABU. Next, let us analyze how each scheme performs in terms of the end-to-end delay between an IoT sensor and its consumer.

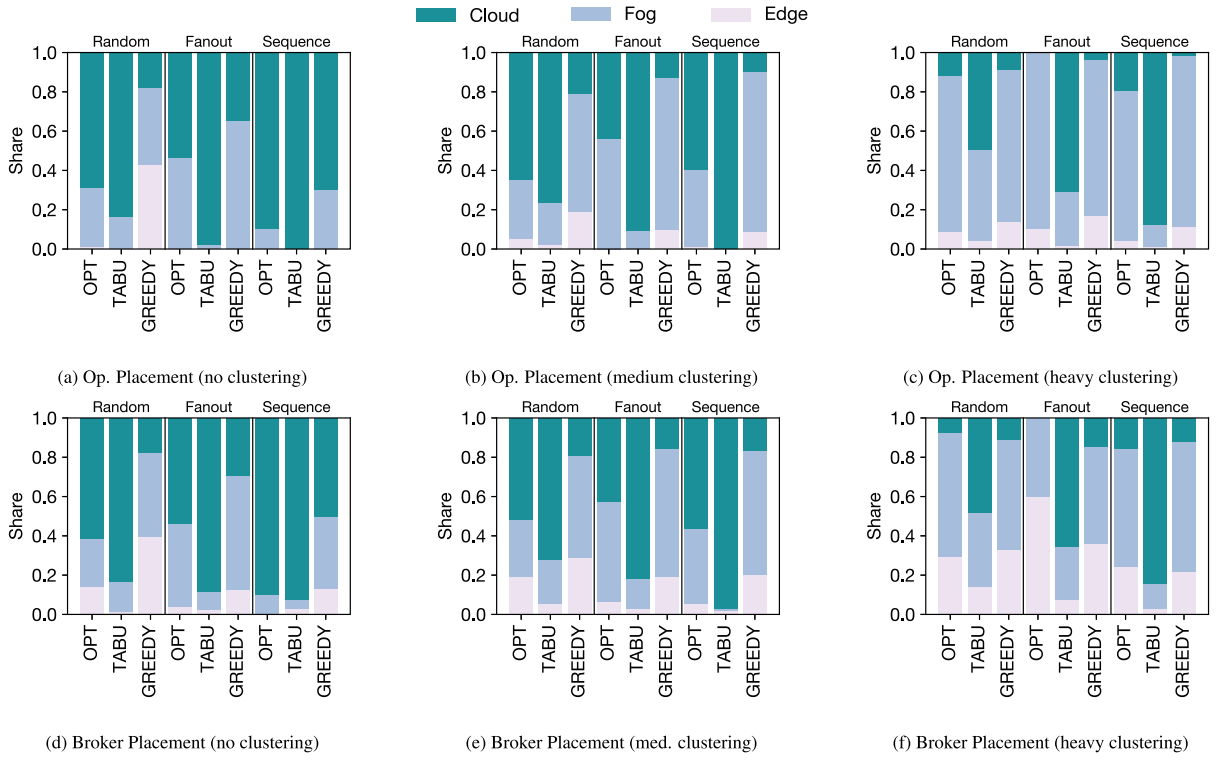
Fig. 5 shows an example of the end-to-end delay distribution of the solutions at the subscribers of the various placement algorithms. The example uses the random application graph topology with heavy clustering, i.e., all publishers and subscribers are on the same node. First, we observe that CLOUD can provide a minimum of approximately 50 ms for its services. This is the expected time to route the traffic to and from the cloud. In CLOUD, the data first has to get to the remote cloud, but once in the cloud, data is processed fast and does not have to be transferred between nodes until it has to be delivered to a subscriber. For applications with strict delay guarantees, CLOUD might fail to deliver a satisfactory user experience. For example, if the latency of rendered image is higher than 15 ms, it might result in motion sickness for augmented reality or virtual reality applications [16]. Moreover, generally speaking, CLOUD is inferior to TABU which maintains delay values in the proximity of those achieved by OPT. Finally, we observe that GREEDY achieves short delays (almost 20% below 50 ms) but might also result in a significantly larger delay for some subscribers as compared to TABU and CLOUD. For example, we record a maximum delay of approximately 170 ms for GREEDY whereas the maximum delay is 150 ms and 120 ms for CLOUD and TABU, respectively. A closer look to our results also shows that GREEDY has to resort to placing operators and brokers farther apart if nearby nodes do not have sufficient capacity.

Next, we quantify the suboptimality of each approach under the aforementioned application topologies and with increasing degree of clustering. Let us first consider the case where the publishers and the subscribers are distributed uniformly, i.e., there is *no clustering*. Fig. 6(a) shows that the deployment in the cloud is close to optimal. For a sequence of operators, it actually is one of the optimal placements as we observe an optimality gap factor of 1. For random and fanout application topologies, CLOUD's



(a) Optimality Gap (no clustering) (b) Optimality Gap (medium clustering) (c) Optimality Gap (heavy clustering)

Fig. 6. Optimality gap achieved by each scheme on the considered application topologies and with three levels of clustering.



(a) Op. Placement (no clustering)

(b) Op. Placement (medium clustering)

(c) Op. Placement (heavy clustering)

(d) Broker Placement (no clustering)

(e) Broker Placement (med. clustering)

(f) Broker Placement (heavy clustering)

Fig. 7. Evaluation results: Subfig. (c)–(e) show the optimality gap, (f)–(h) the operator placement locations and (i)–(k) the locations of brokers.

optimality gap factor is below 1.2. Therefore, we can articulate in these cases that CLOUD can be the preferred approach for broker and operator placement due to its simplicity and near-optimal performance. Comparing TABU with CLOUD, TABU does not improve the gap significantly over CLOUD, despite being a more advanced approach compared to the former. While GREEDY sometimes yields the optimal placement, it usually shows an optimality gap of 1.2 to 1.4, with a worst case of about 1.8.

For medium clustering, we observe that the gap between the CLOUD and TABU increases, meaning that the performance difference between these heuristics and OPT is higher compared to the no clustering case. More specifically, the optimality gap of CLOUD and TABU varies from 1 to 1.5 for random topology whereas it can be as high as 2 for the fanout application graph. For instance, for the fanout scenario, comparing TABU and CLOUD, we observe that TABU outperforms CLOUD on average for fanout topology and variance of its performance is lower as compared to CLOUD. On a sequence topology, performance difference of TABU and CLOUD is indistinguishable. Finally, Fig. 6(c) depicts the case of high clustering which corroborates our conclusions from the previous case: higher optimality gap and better performance of TABU over CLOUD. To summarize, if a large spatial distribution of publishers and subscribers is expected, CLOUD is a fairly good option. But, with increased clustering, there is room for further improvements by either a tabu-based approach or an optimizer.

Now, we examine placement decisions of each approach by taking a closer look to the utilization of each layer for operators and also for brokers. In Figs. 7(a), 7(b), and 7(c), we show the fraction of operators placed in the cloud, fog, and edge. Please note that publishers and subscribers are excluded from these plots as their locations are outside the control of our placement algorithms. In agreement with the previous observations on the impact of clustering, we observe that with higher clustering, the cloud layer becomes less preferable; more operators are placed outside the cloud. More specifically, we observe that the fog layer starts to host an increasing fraction of the operators while the edge layer hosts only a small fraction. We attribute this behavior to the higher resource availability at the fog layer. Interestingly, TABU usually prefers a cloud placement while OPT might exhibit a different utilization pattern. To give a comprehensive example, for the fanout application graph in Fig. 7(c), OPT almost never utilizes the cloud for the operators whereas TABU places more than 70% of the operators to the cloud. Nevertheless, as Fig. 6(c) shows, TABU outperforms CLOUD in terms of optimality gap factor. As a final note, although layer utilization behavior of GREEDY resembles that of OPT, its performance is much lower compared to TABU. From these two observations, therefore, we conclude that while utilizing the right layer is important to achieve a near-optimal performance, placing the operators on the right nodes has a more significant impact on the performance.

Finally, Figs. 7(d), 7(e), and 7(f) show the utilization of each layer by the brokers. While they are in general similar, there are more brokers on the edge, which are presumably the brokers mainly for publishers and subscribers. Since we do not fix any brokers in this scenario, all brokers are included in the plot, while their corresponding operators might be excluded from the previous plots. The results of the broker placement simulations are also in agreement with our earlier conclusions: with a higher degree of spatial locality of publishers and subscribers, the cloud becomes less preferable also for the brokers and the fog/edge layers start to host an increasing number of brokers under OPT.

5.5. A realistic use case: smart crosswalk scenario

To further motivate possible use cases and the applicability of the optimal and heuristic solutions, we consider a small real-world application as an example. The scenario and application is derived from the finding in [17] and illustrated in Fig. 8. The example application is a smart crosswalk that uses traffic camera analytics to detect pedestrians in a wheelchair and provide additional time when needed to let the pedestrians to safely cross the street. In this scenario, a video camera that overlooks the

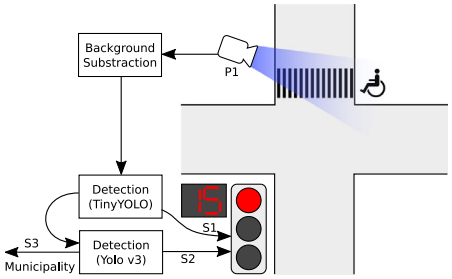


Fig. 8. Example of a smart crosswalk application and the corresponding video analytics pipeline.

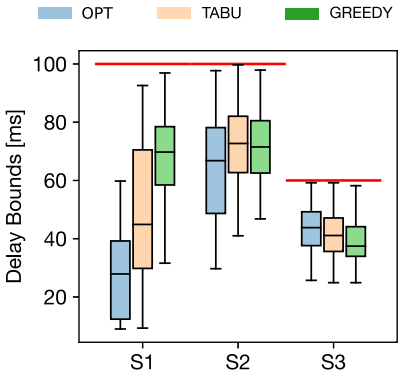


Fig. 10. Achieved Latency of Subscribers S1 and S2 (Traffic Light), as well as S3 (Municipality Backend) of the placement in comparison to the bounds set as a parameter (red line).

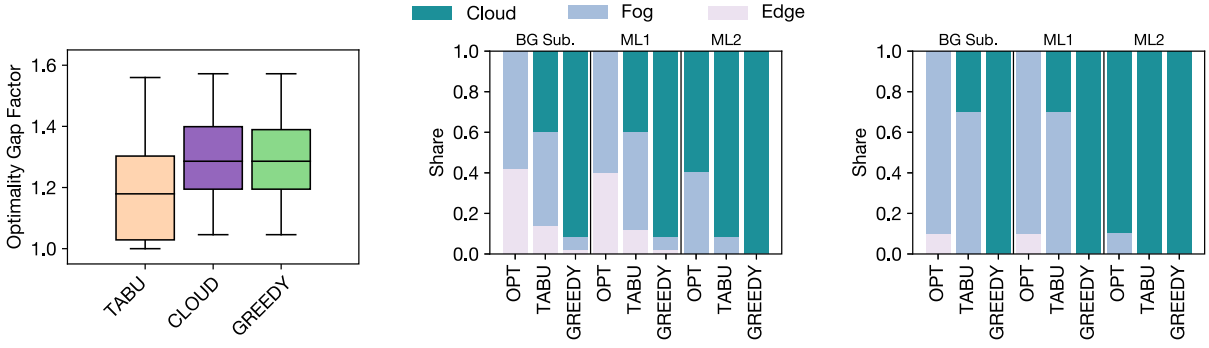
crosswalk is the publishing sensor. The video pipeline consists of three operators: (1) the background subtraction operator extracts foreground objects in each frame; (2) a light-weight deep neural network (DNN), such as TinyYOLO [18] is used to detect crossing objects, such as pedestrians, bicycles, or pedestrians in wheelchairs; (3) if the confidence of the light-weight DNN is not high enough, a more sophisticated DNN, such as YOLO v3 [18], is executed. As subscribers, we consider three subscribers: (S1) the (local) traffic light itself, (S2) a centralized (cloud-based) subscriber and (S3) municipality backend that gathers various statistics of its smart city to improve its services to the citizens.

In the example, each subscriber has realistic delay bounds associated with it. We assume that the smart traffic light can tolerate a delay of 100 ms. Since the municipality might need to act as a central controller that instructs other traffic lights across the city, we assume that its delay bounds are only 60 ms. As topologies, we consider the same 40 node topologies introduced earlier. However, we assume that accelerating hardware, such as CUDA and OpenCV enabled graphics cards, are available throughout the topology the municipality deems suitable for a deployment. We run 5 applications on 10 topologies for a total of 50 runs.

The results of the placement with the optimal solution (OPT) and the TABU, GREEDY, and CLOUD heuristics are shown in Fig. 9. Fig. 9(a) shows the optimality gap achieved for each of the heuristics while Figs. 9(b) and 9(c) show the location of operators and the location of brokers, respectively. In these figures, BG Sub. stands for background subtraction while ML1 and ML2 stand for the machine-learning detection stages, i.e., TinyYOLO and Yolo v3. We observe that the GREEDY algorithm results in a very similar deployment to the CLOUD heuristic, i.e. the GREEDY places almost all ($>$ 90%) operators and all brokers in the cloud. As expected, the optimality gap is also very similar between those two heuristics. The TABU algorithm improves those placements by placing more operators towards the fog and edge. In particular, about 10% of the background subtraction operators as well as ML1 operators are run locally on the edge, while around 50% are run on the fog. The heavier detection operator ML2 is mostly ($>$ 90%) run in the cloud.

The optimal solution, however, is to never place the operators for background subtraction and ML1 in the cloud. For both cases, the operators are placed on the edge about 40% of all runs and on the fog about 60%. About 40% of ML operators are placed on fog nodes. We also observe that the broker placement is not just a trivial collocation of brokers with operators. Fig. 9(c) clearly shows that the output brokers of the three pipeline steps are usually more centrally located then the operators themselves. However, the optimal location for brokers for background subtraction and ML1 is never in the cloud: For about 90% of cases the optimal location is in the fog, while for the remaining cases it is even at the edge. This result confirms our argument that broker placement is not trivial and is an important consideration when deploying interconnected operators.

Fig. 10 shows the delay bound for the subscribers S1 and S2 (traffic lights), and S3 (municipality backend) that are specified as red lines. Since the optimal solution as well as the heuristics consider delay bounds to be strict requirements, i.e., deployments not meeting the deadlines are considered infeasible, we see that all the approaches indeed meet the deadlines in every run. Another insight that we can gain is that TABU and OPT essentially do not significantly improve the performance for S2 or S3, but improve the latency for S1 which is located close to the publisher.



(a) Optimality Gap (b) Operator Placement (c) Broker Placement

Fig. 9. Results of the placement of the smart crosswalk video pipeline placement: Subfig. (a) shows the optimality gap of the heuristics, (b) and (c) show the location of operators and brokers respectively.

This realistic example shows that the optimal solution as well as the heuristics that are presented in this work can indeed be used for realistic non-trivial use cases. It also shows that delay bounds can be specified and will be respected by all approaches. The results also highlight that the placement of brokers is not immediately obvious based on the placement of operators and hence the placement of brokers is an important consideration. Finally, the results again show that placing some operators and brokers closer to the edge will significantly improve the latency for local control loops.

5.6. Discussion on deployment cost

In the previous sections, our focus was on the metrics affecting the application performance such as delay. However, from the application owner's viewpoint, monetary cost of a deployment is an important factor affecting its deployment decision. Hence, we present in this section some observations regarding the monetary costs of the deployment each approach yields.

While the price of cloud computing resources can be quite accurately modeled based on the pricing of established providers, we lack realistic pricing models for edge and fog services as still there is no consensus on how these infrastructures will be deployed, e.g., by independent edge providers [19] or by cloud providers. While we argue that, due to economies of scale, edge resources will not reach the low price point of resources in dedicated datacenters in the foreseeable future, a plausible counter-argument is that resources at the edge are currently available but not fully utilized. Hence, those resources could be significantly cheaper or even considered free for some operators.

We therefore consider the two extremes: (i) the cloud is cheaper than fog and edge resources and (ii) the resources at the edge are significantly cheaper. We study the average cost of deployment estimated as a unit-less value roughly based on today's cloud prices in US dollars per month. As an example, we show the results for the heavy clustering case in Fig. 11. Fig. 11(a) shows the case of fog resources being expensive compared to the cloud resources; Fig. 11(b) presents the results for the case of cheaper fog and edge resources.

We observe the effect that OPT and GREEDY deploy operators closer to the edge, making the overall cost higher in the first case and lower in the second. While the cost penalty in the first scenario is between 20% and 30%, when fog and edge resources are cheap, an optimal deployment with regard to the performance can actually lead to almost 50% cost savings in the case of a fanout application topology. We also see that the greedy algorithm, while not optimal in the performance metrics analyzed previously, will result on average in cheaper deployments than the optimal solution if fog and edge resources are cheap. While TABU improves the cloud only solution in utility, it is only slightly more expensive when the cloud is cheap, making it a potentially worthwhile trade-off.

5.7. Discussion on a practical system

Since our system assumes a certain knowledge about the network topology and the application graphs, we discuss here how to estimate those values in a practical system. Our system heavily depends on the knowledge of the latency and available bandwidth between each node. Those values can trivially be measured between each node pair, although the number of measurements required would become prohibitively large for larger topologies. However, the Vivaldi algorithm [20] can be used to estimate the topology without measuring each pair individually. The basic idea is that each node is positioned w.r.t. other nodes in a two dimensional plain based on their observed latency. The distance in this plain can then be used as an estimator for the real-world delay. Additionally, the CPU and memory resources can be derived by running the passmark benchmark on every node. This will yield some unitless score for CPU and the amount of free memory the system has. Operators can be run on a reference system and can be related to the amount of total processing power the system has. This way, a similar minimal passmark score for each operator can be derived. Of course, those resources can only serve as estimates.

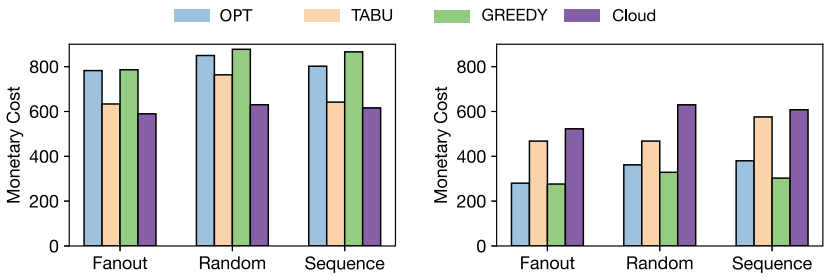
6. Related work

We can categorize the related work into three groups: *IoT analytics deployment*, *broker-based IoT*, and *mobile edge computing* in an IoT setting.

IoT analytics deployment: The most related work to ours is [10] which focuses on optimal placement of IoT analytics functions on edge, fog, or cloud nodes to maximize the number of satisfied IoT analytics requests. As the optimal placement problem is computationally hard, [10] proposes a heuristic which starts placing operators with the lowest demand for the scarcest resource so that number of satisfied applications can be maximized. In a similar setting, Hong et al. [21] start with the request consisting of the lowest number of operators (modules) and assigns its modules to the device with the least popularity, e.g., number of requests preferring this device. Module deployment algorithm (MDA) of [21] has the same objective as [10], i.e., maximizing number of satisfied requests, but overlooks many practical issues such as capacity limitations of each link. DROPLET [22] aims at minimizing the total completion time of an application consisting of multiple operators with a certain dependency. To achieve this goal, DROPLET models the execution time of each operator on different resources, e.g., containers, as well as communication cost between the dependent operators using the link capacities connecting the corresponding resources hosting these operators. The key idea of DROPLET is to exploit the possibility of running operators in parallel (even those with dependency relation on different data chunks) so that the completion time of an application can be decreased.

Our paper differs from all these works in that we consider a broker-based communication and jointly deploy brokers and operators while [10,21] assume broker-less communication. Broker-based IoT solutions offer better scalability over brokerless peer-to-peer communications of the publisher and the subscriber [9]. In broker-less design, compute-constrained sensors might suffer from high latency in serving their subscribers [9]. On the contrary, broker-based design eliminates the direct communication between a sensor and its subscribers, which consequently reduces the sensor's energy consumption. This is especially desirable for resource constrained sensors.

The closest work to ours is FogTorch [12,23,24]. In [12], authors propose a general model to support QoS-aware deployment of IoT applications over Fog infrastructures. They also design a greedy algorithm for solving the proposed model and develop a prototype which is later extended to FogTorchΠ in [23] that takes also the financial costs into account. In [24], the authors extend their work with the security requirements of the applications. While these proposals are very valuable, for the particular problem of *joint* broker and operator placement they are less expressive than the model presented in this work. Moreover, for determining the order of nodes in each greedy step, FogTorch uses a cost function that does not reflect the expected latency or other performance metrics. FogTorch puts its focus on finding deployments that are feasible in terms of available hardware



(a) Fog expensive compared to cloud

(b) Fog cheap compared to cloud

Fig. 11. Average deployment cost in a unit based on today's US dollar per month: Fig. 11(a) shows the case of fog resources being expensive compared to cloud resources; Fig. 11(b) presents the results for the case of cheaper fog and edge resources.

and software on the nodes, rather than the performance, as our solution does.

Broker-based IoT: New challenges and questions emerge with the introduction of brokers, such as balancing the load on the brokers [11] or functionality a broker should embody [11,25]. FogMQ [9,26] designs a middleware to let each broker (device clone) migrate autonomously from one hosting cloud to another depending on the perceived latency and broadcasted statistics of each cloud. [11] proposes to cache IoT sensor data at brokers for decreasing the energy consumption of the IoT servers, which are highly likely battery operated. Authors decide on whether to cache an IoT resource and in case of caching decision which broker to cache with a goal of minimizing total latency of brokers serving the incoming requests. Our problem is also similar to [11] in that we aim at finding the brokers which should act as the responsible broker for an IoT sensor resource. However, different than [11], we do not assume a given broker infrastructure and instead determine which network location should host a broker.

Mobile edge computing (MEC): MEC aims at enabling cloud capabilities and services closer to the user, by pushing computational and storage resources towards the edge. An overview and a research outlook on MEC is provided in [27]. They summarized modeling methodologies and key components, as well as give potential research directions and recent standardization efforts. Another survey is provided in [28]. In [29], the authors propose an efficient one-dimensional search algorithm that finds an optimal task scheduling policy in the context of MEC. They show that their stochastic task scheduling policy achieves the minimum average delay in various scenarios. The authors of [30] try to solve the problem of mapping an incoming stream of application graphs onto a physical graph.

7. Conclusions & future work

The placement of operators and brokers in a diverse network of cloud, fog, and edge nodes is an important problem for the success of future IoT systems. In this work, we formulated the problem of joint deployment of brokers and operators on the existing edge, fog, and cloud resources as an optimization problem considering the locations of the IoT sensors and the consumers (e.g., IoT application users) that use the data published by these sensors. Since the formulated problem belongs to the family of computationally-hard problems, we proposed heuristic solutions for this joint problem.

Our simulation results reveal that the optimal deployment heavily depends on the distribution or clustering of publishers and subscribers. In case of a wide spatial distribution of publishers and subscribers, the deployment of broker and operators in the cloud is close to optimal for most cases. With more clustering of publishers and subscribers, there is an increasing opportunity to improve the deployment in the cloud. We showed that the tabu algorithm can meaningfully improve on already-obtained solutions of other heuristics, while a simple greedy algorithm is on average not better than the deployment in the cloud.

As future work, we plan to expand our findings with regard to a more precise and granular definition of the clustering of publishers and subscribers and test the algorithms in a more realistic setting. Given that network state changes over time, solving the placement problem periodically by considering the possible changes in the network is another interesting direction to investigate.

CRediT authorship contribution statement

Daniel Happ: Conceptualization, Methodology, Software, Writing - original draft, Writing - review & editing, Visualization, Formal analysis. **Suzan Bayhan:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing, Formal analysis. **Vlado Handziski:** Conceptualization, Methodology, Writing - original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) within the project SecureFog under the grant number 16KIS0777.

References

[1] L. Yu, N. Wang, X. Meng, Real-time forest fire detection with wireless sensor networks, in: Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005, Vol. 2, 2005, pp. 1214–1217.

[2] G. Ananthanarayanan, V. Bahl, L. Cox, A. Crown, S. Nogbahi, Y. Shu, Video analytics - killer app for edge computing, in: ACM MobiSys, in: MobiSys, 2019, pp. 695–696.

[3] M. Szustakowski, W.M. Ciurapinski, M. Zyczkowski, Trends in opto-electronic perimeter security sensors, in: E.M. Carapezza (Ed.), Unmanned/Unattended Sensors and Sensor Networks IV, 6736, International Society for Optics and Photonics, SPIE, 2007, pp. 215–226, http://dx.doi.org/10.1117/12.737733.

[4] P. Michalák, P. Watson, PATH2iot: A holistic, distributed stream processing system, in: 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2017, pp. 25–32.

[5] B. Zhang, N. Mor, J. Kolb, D.S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, J. Kubiatowicz, The cloud is not enough: Saving IoT from the cloud, in: USENIX Workshop on Hot Topics in Cloud Computing (HotCloud), 2015.

[6] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.
[7] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.
[8] D. Happ, N. Karowski, T. Menzel, V. Handziski, A. Wolisz, Meeting IoT platform requirements with open pub/sub solutions, Ann. Telecommun. 72 (1) (2017) 41–52, http://dx.doi.org/10.1007/s12243-016-0537-4.
[9] S. Abdelwahab, B. Hamdaoui, Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms, 2016, arXiv preprint arXiv:1610.00620.
[10] H.J. Hong, P.H. Tsai, A.C. Cheng, M.Y.S. Uddin, N. Venkatasubramanian, C.H. Hsu, Supporting internet-of-things analytics in a fog computing platform, in: IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2017, pp. 138–145.
[11] X. Sun, N. Ansari, Traffic load balancing among brokers at the IoT application layer, IEEE Trans. Netw. Serv. Manag. (2017).
[12] A. Brogi, S. Forti, QoS-aware deployment of IoT applications through the fog, IEEE Internet Things J. 4 (5) (2017) 1185–1192.
[13] R. Mijumbi, J. Serrat, J.L. Gorricho, N. Bouten, F.D. Turck, S. Davy, Design and evaluation of algorithms for mapping and scheduling of virtual network functions, in: IEEE Conference on Network Softwarization (NetSoft), 2015.
[14] Gurobi Optimization, LLC, Gurobi optimizer reference manual, 2019, http://www.gurobi.com.
[15] D. Happ, S. Bayhan, On the impact of clustering for IoT analytics and message broker placement across cloud and edge, in: 15th ACM European Conference on Computer Systems (EuroSys 2020), 3rd ACM International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2020), ACM, Irákleion, Greece, 2020, http://dx.doi.org/10.1145/3378679.3394538.
[16] M.S. Elbamby, C. Perfecto, M. Bennis, K. Doppler, Toward low-latency and ultra-reliable virtual reality, IEEE Netw. 32 (2) (2018) 78–84.
[17] G. Ananthanarayanan, V. Bahl, L. Cox, A. Crown, S. Nogbahi, Y. Shu, Video analytics – killer app for edge computing, in: Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, 2019, pp. 695–696.
[18] J. Redmon, A. Farhadi, Yolov3: An incremental improvement. arxiv 2018, 2018, pp. 1–6, arXiv preprint arXiv:1804.02767.
[19] A. Zavodovski, N. Mohan, S. Bayhan, W. Wong, J. Kangasharju, ExEC: Elastic extensible edge cloud, in: Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking, 2019, pp. 24–29.
[20] F. Dabek, R. Cox, F. Kaashoek, R. Morris, Vivaldi: A decentralized network coordinate system, in: ACM SIGCOMM Computer Communication Review, 34, (4) 2004, pp. 15–26.
[21] H.J. Hong, P.H. Tsai, C.H. Hsu, Dynamic module deployment in a fog computing platform, in: 18th Asia-Pacific Network Operations and Management Symposium, APNOMS, 2016, pp. 1–6, http://dx.doi.org/10.1109/APNOMS.2016.7737202.
[22] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, G. Agha, DROPLET: Distributed operator placement for IoT applications spanning edge and cloud resources, in: IEEE International Conference on Cloud Computing, 2018, pp. 1–8.
[23] A. Brogi, S. Forti, A. Ibrahim, Predictive analysis to support fog application deployment, in: Fog and Edge Computing, John Wiley & Sons, Ltd, 2019, pp. 191–221, http://dx.doi.org/10.1002/9781119525080.ch9.
[24] S. Forti, G.-L. Ferrari, A. Brogi, Secure cloud-edge deployments, with trust, Future Gener. Comput. Syst. 102 (2020) 775–788.
[25] B. Karakostas, N. Bessis, Intelligent brokers in an internet of things for logistics, in: Proceedings of the International Conference on IoT and Cloud Computing, ACM, 2016, p. 5.
[26] S. Abdelwahab, B. Hamdaoui, Flocking virtual machines in quest for responsive IoT cloud services, in: IEEE International Conference on Communications (ICC), 2017, pp. 1–6.
[27] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.
[28] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1628–1656.
[29] J. Liu, Y. Mao, J. Zhang, K.B. Letaief, Delay-optimal computation task scheduling for mobile-edge computing systems, in: IEEE International Symposium on Information Theory, ISIT, IEEE, 2016, pp. 1451–1455.
[30] S. Wang, M. Zafer, K.K. Leung, Online placement of multi-component applications in edge computing environments, IEEE Access 5 (2017) 2514–2533.

Daniel Happ received his M.Sc. degree in computer science in 2013 from Free University of Berlin, Germany. He has since been a Ph.D. student at the Telecommunication Networks Group (TKN) at Technische Universität Berlin. His research interests include cloud-based IoT platforms, message-oriented middleware, pub/sub systems and fog computing.

Suzan Bayhan is an Assistant Professor at the University of Twente, the Netherlands and an Adjunct Professor (Docent) at the University of Helsinki, Finland. Suzan earned her Ph.D. in Computer Engineering in 2012 from Bogazici University, Turkey and worked at the University of Helsinki, Aalto University, and TU Berlin between 2012–2019. She was a short-term visitor at Princeton University EdgeLab and Aalto University. She was on N2Women Board as mentoring co-chair during 2017–2018. Her current research interests include spectrum sharing and coexistence of wireless networks, WiFi and LTE radio resource management, and edge computing.

Vlado Handziski received his Ph.D. degree (summa cum laude) in electrical engineering from Technische Universität Berlin (TU Berlin), in 2011. Until 2020 he was a Senior Researcher with the Telecommunication Networks Group, TU Berlin, coordinating the research and teaching activities in the areas of networked embedded systems, Internet of Things, and cyber–physical systems. In 2015 and 2017, he was an Adjunct/Interim Professor at the Chair for Embedded Systems, TU Dresden. He has participated and led research activities in more than ten research projects with funding at European, national, and regional levels, and has contributed to international standardization activities in the area of evaluation of indoor localization systems. He is currently the Chief Technical Officer of R3 Reliable Realtime Radio continuing research and development in the area of WiFi-based URLLC systems for industrial automation.