Security- and privacy-aware IoT application placement and user assignment*

Zoltán Ádám Mann

University of Amsterdam, Amsterdam, The Netherlands

Abstract. Applications for the Internet of Things (IoT) may use, beyond the IoT devices themselves, also edge and cloud resources. Thus, the modules of an application can be placed on a variety of nodes with different capabilities in terms of security, trustworthiness, and capacity. Application modules may exist in multiple instances. This makes it possible to assign users to the most appropriate module instances, taking into account requirements on security, privacy, and latency. There is a non-trivial interplay between application placement decisions and user assignment decisions. For example, if a certain user is assigned to a module, then that module may not be allowed to be placed on nodes not trusted by the user. However, most existing research neglects this interplay and its implications on security and privacy. In this paper, we address the joint problem of application placement and user assignment. Beside capacity and latency constraints, we consider several types of security and privacy constraints: (i) module-level location constraints, (ii) user-level location constraints, (iii) co-location constraints, and (iv) k-anonymity constraints. We formalize the problem and develop an algorithm to solve it using quadratically constrained mixed integer programming. We demonstrate the applicability of the proposed approach by applying it to an IoT system in the smart home domain. Controlled experiments on problem instances of increasing size show that the algorithm can solve even large problem instances in acceptable time.

Keywords: Internet of Things · IoT · Fog computing · Edge computing · Application placement · Security · Privacy

1 Introduction

Modern computing infrastructures offer a continuum of computational resources, from cloud data centers through fog and edge nodes to end devices in the Internet of Things [4]. Network connections among the different compute nodes make it possible to place the modules of an application on different nodes. Taking into account the different capacity of the nodes and the network latency between nodes, optimal decisions on application placement can be made [5]. However, it is also important to take into account the heterogeneity of the nodes in terms of

* Published in *Computer Security — ESORICS 2021 International Workshops*, Lecture Notes in Computer Science, vol. 13106, Springer, pp. 296-316, 2022

security and privacy protection: for example, cloud data centers may offer better security than fog nodes, whereas processing data from end devices in nearby fog nodes may be more advantageous from a privacy point of view than offloading to the cloud [2, 22, 3].

To optimally serve users, a distributed application may contain multiple instances of the same module. This is beneficial for example for a geographically dispersed user base: the different instances of a module can be placed in such a way that all users can access a nearby instance with low latency. Beside the latency implication, the assignment of users to module instances is also important from a data protection point of view. For example, a module may have to have at least k users assigned to guarantee k-anonymity [24, 15].

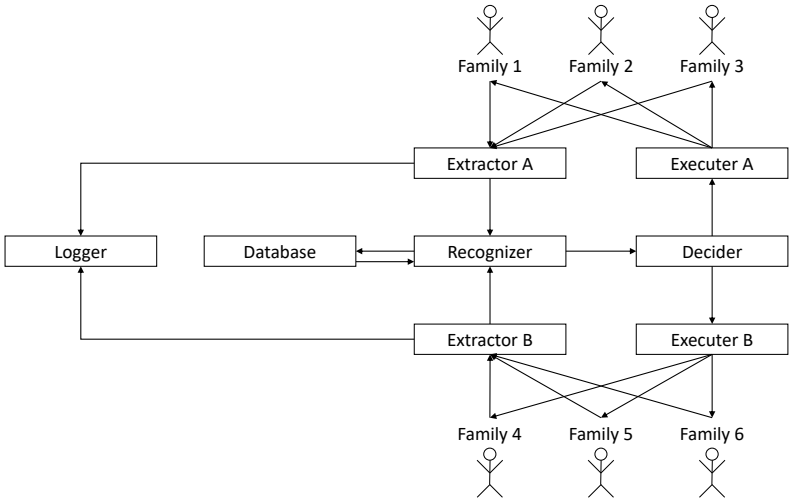
Decisions on application placement (i.e., which application module to place on which infrastructure node) and decisions on user assignment (i.e., which user to assign to which module instance) may mutually impact each other. For example, for data protection reasons it may not be allowed to process the data of certain users in certain locations. Hence, if such a user is assigned to a module, then that module is not allowed to be placed on nodes that are in the forbidden locations. Moreover, if many users are assigned to a module, this may lead to an increase in the computational needs of the module, thus requiring a node with high computational capacity.

Despite this interplay between application placement and user assignment, most existing work in this domain targets either application placement [6] or user assignment [10], but not both. Furthermore, most related work either completely ignores security and privacy requirements, or handle them in a rudimentary way.

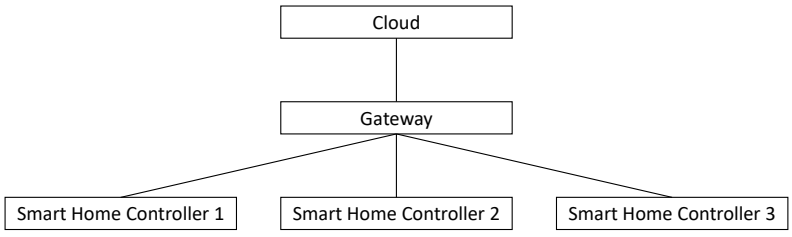
To the best of our knowledge, this is the first paper to address the joint problem of application placement and user assignment with a focus on security and privacy requirements. Beside capacity and latency constraints, we investigate four types of constraints that result from security and privacy requirements:

- Module-level location constraints, which prohibit the placement of certain critical application modules on certain insecure infrastructure nodes
- User-level location constraints, which prohibit the placement of modules processing the data of certain users on infrastructure nodes not trusted by those users
- Co-location constraints, which prohibit the placement of certain pairs of application modules on the same infrastructure node (e.g., because such co-location could lead to side-channel attacks)
- k-anonymity constraints, which ensure that the data pertaining to at least k users is processed together

We formalize the resulting problem, which combines application placement and user assignment, also taking into account the mentioned types of security and privacy constraints. We devise a quadratically constrained mixed integer formulation, which can be solved by an appropriate solver. The resulting algorithm is guaranteed to always find a placement of the application modules and an assignment of the users that satisfy all constraints, whenever this is possible. The practical applicability of the proposed approach is shown by applying it to



(a) Software application



(b) Infrastructure

Fig. 1. The Smart Bell example

a fog computing case study from the smart home domain. Moreover, we perform controlled experiments on problem instances of increasing size to assess the scalability of the approach. The results show that the algorithm can solve even quite large problem instances in acceptable time on a commodity computer.

2 A motivating example

We consider an example IoT system from the smart home domain, based on the Smart Bell system presented in [28]. The aim of Smart Bell is to recognize visitors of a set of smart homes and react intelligently. For this purpose, cameras capture images of visitors. The captured images are compared to images stored in a database. This way, the system can recognize inhabitants, their friends, neighbors, and neighbors' friends. On this basis, the system can automatically open the door, notify the inhabitants of the home, or activate an alarm.

The Smart Bell system serves a block of smart homes that are connected via a network. The Smart Bell software application is shown in Fig. 1(a). The

rectangles in the figure are the modules of the application, the arrows represent data flows. Extractor modules are used to extract faces from pictures taken by the cameras. The Recognizer compares the extracted face with the faces in the Database in order to recognize known faces. Based on the result of the Recognizer, the Decider decides how to react, and the reaction is carried out by the Executer. The Logger logs the visits in an anonymized form.

The stick figures represent users together with their end devices. In our case, a family, together with the smart end devices in their home (camera, door control, alarm) is represented by a stick figure, and is referred to as a user in the following.

To serve several users, some modules may exist in multiple instances. In the example of Fig. 1(a), there are two instances of the Extractor and Executer modules. Thus, users (in this case, the families whose homes are to be served) can be assigned to one of two alternative *data processing path*s:

1. User → Extractor **A** → Recognizer → Database → Recognizer → Decider → Executer **A** → User
2. User → Extractor **B** → Recognizer → Database → Recognizer → Decider → Executer **B** → User

In the shown example, Families 1–3 are assigned to the first, and Families 4–6 to the second data processing path. However, this assignment is not predefined; also other assignments would be possible.

The infrastructure available for the Smart Bell system is shown in Fig. 1(b), consisting of nodes and links. Different nodes may have different computational capacity: the capacity of the Smart Home Controllers is very limited, while the Gateway offers higher capacity, and the capacity of the Cloud is practically unlimited. On the other hand, the link between the Cloud and the Gateway is characterized by a much higher latency than the links between the Gateway and the Smart Home Controllers. Each module of the application has to be placed onto one of the infrastructure nodes. It has to be ensured for each infrastructure node that its computational capacity is not exceeded by the total computational load of the modules that the node should host.

An application processing private or otherwise sensitive data may have to satisfy different types of security and privacy requirements. In our case, for reasons of privacy protection, the Database is not allowed to be placed in the Cloud (module-level location constraint), since the Database contains sensitive personal information and placing it in the Cloud would potentially enable unauthorized parties to gain access to that sensitive information. Smart Home Controller 1 is mounted in the home of Family 1, and Family 2 does not trust Family 1, so that modules processing data of Family 2 must not be placed on Smart Home Controller 1 (user-level location constraint). Executer A and B must not be placed on the same node, so that at least one of them works even in the case of the failure of a node and can provide backup for the other instance (co-location constraint). Both Extractor A and B must be assigned at least three users each so that the required level of anonymity can be guaranteed (k-anonymity constraint).

3 Problem formulation

In this section, we formalize the combined problem of application placement and user assignment, also taking into account security and privacy constraints. For this purpose, we describe the inputs of the problem, the output that needs to be computed, and the constraints that the output must fulfill. The used notation is summarized in Table 1.

3.1 Inputs

The inputs to the addressed problem comprise the description of the infrastructure, of the applications to deploy, and of the users to serve, as well as further information needed for formulating the security and privacy requirements.

Infrastructure. The set of infrastructure nodes (servers, data centers etc.) is denoted by N. Each node $n \in N$ is characterized by its computational capacity $C_n \in \mathbb{R}_{\geq 0}$. For each pair of nodes $n_1, n_2 \in N$, the network connection between them is characterized by a latency value $\ell_{n_1,n_2} \in \mathbb{R}_{\geq 0}$.

Applications. The set of applications to deploy is denoted by A. Each application $a \in A$ consists of a set of modules M_a. The set of all modules of all applications is $M = \bigcup\{M_a : a \in A\}$. For a module $m \in M$, its size (i.e., the computational capacity required by the module) is given by

$$S_m(w_m) = \alpha_m + \beta_m \cdot w_m,$$

where $\alpha_m \in \mathbb{R}_{\geq 0}$ and $\beta_m \in \mathbb{R}_{\geq 0}$ are given constants and $w_m \in \mathbb{N}$ is the number of users served by module m. Note that only α_m and β_m are part of the input, w_m is not; hence, the input defines the function $S_m(\cdot)$.

In an application $a \in A$, a data processing path P is a sequence $(m_1, m_2, \ldots, m_{\kappa_P})$, where $m_i \in M_a \cup \{*\}$ for all $1 \leq i \leq \kappa_P$. Here, $*$ is a symbol representing a user. An element of the sequence (either a module or the symbol $*$) can appear multiple times in a data processing path. We write $m \in P$ if module m appears at least once in the sequence of the data processing path P. For each application $a \in A$, a set of data processing paths $\mathcal{P}_a$ is given. Moreover, each $P \in \mathcal{P}_a$ is associated with a maximum allowed latency, denoted as L_P. The set of all data processing paths of all applications is $\mathcal{P} = \bigcup\{\mathcal{P}_a : a \in A\}$. For a module $m \in M$, the set of data processing paths containing m is $\mathcal{P}(m) = \{P \in \mathcal{P} : m \in P\}$.

Users. A finite set U of users is given. For each user $u \in U$, the location of the user in the network is given as $n_u \in N$. Moreover, for each user $u \in U$, the application that u wants to use is given as $a_u \in A$.

Information for security and privacy requirements. For formulating the security and privacy requirements, some further notation is necessary. For a module $m \in M$, $I_m \subset N$ denotes the set of illegal nodes for m, i.e., the nodes that are not allowed to host m. In addition, for a user $u \in U$ and a module $m \in M_{a_u}$, $I_{m,u} \subset N$ denotes the set of illegal nodes for the pair (m, u), i.e., the nodes that are not allowed to host m if m processes data of user u.

The set $\mathcal{I}$ consists of pairs of modules that must not be colocated. If $(m_1, m_2) \in \mathcal{I}$, then the modules m_1 and m_2 must not be placed on the same node.

Table 1. Notation overview

Notation	Description
Inputs	
N	Set of all infrastructure nodes
C_n	Computational capacity of node n
ℓ_{n_1,n_2}	Latency between nodes n_1 and n_2
A	Set of applications to deploy
M_a	Set of modules of application a
M	Set of modules of all applications
α_m, β_m	Parameters in the function $S_m(\cdot)$
$\mathcal{P}_a$	Set of data processing paths in application a
$\mathcal{P}$	Set of all data processing paths of all applications
$\mathcal{P}(m)$	Set of data processing paths containing module m
κ_P	Length of data processing path P
$*$	Symbol representing a user in a data processing path
L_P	Maximum allowed latency of data processing path P
U	Set of users
n_u	Location (i.e., node) of user u
a_u	Application that user u wants to use
I_m	Set of illegal nodes for module m
$I_{m,u}$	Set of illegal nodes for module m with data of user u
$\mathcal{I}$	Set of pairs of modules that must not be colocated
k_m	Minimum number of users for module m
Outputs	
$f(m)$	Node on which module m is placed
$f^{-1}(n)$	Set of modules placed on node n
$g(u)$	Data processing path to which user u is assigned
$g^{-1}(P)$	Set of users assigned to data processing path P
Other	
w_m	Number of users served by module m
$S_m(w_m)$	Computational capacity required by module m
$L(u)$	Latency perceived by user u

For a module $m \in M$, $k_m \in \mathbb{N}$ denotes the minimum number of users that must be assigned to m to achieve a sufficient level of anonymity. ($k_m = 0$ means that there is no such limitation for the given module.)

3.2 Outputs

Our aim is to determine two mappings: the placement of the applications and the assignment of the users.

The placement of the applications is a function $f : M \to N$. For a module $m \in M$, $f(m)$ is the node in N on which m is placed. The inverse function of f is denoted by f^{-1}; for a node $n \in N$, $f^{-1}(n)$ is the set of modules placed on n.

The assignment of the users is a function $g : U \to \mathcal{P}$. For a user $u \in U$, $g(u)$ is the data processing path in $\mathcal{P}$ to which user u is assigned. The inverse function

of g is denoted by g^{-1}; for a data processing path $P \in \mathcal{P}$, $g^{-1}(P)$ is the set of users assigned to P.

The output that we need to determine thus consists of the functions f and g. Based on the function g, we can compute the number of users served by a module $m \in M$ as follows:

$$w_m = \sum_{P \in \mathcal{P}(m)} |g^{-1}(P)|. \tag{1}$$

To see the correctness of (1), it should be noted that each user is assigned to exactly one data processing path, and hence is contained in exactly one of the $g^{-1}(P)$ sets. Therefore, each user served by module m is counted exactly once in (1). Moreover, it should be noted that w_m is not an output, but an auxiliary number depending on the function g and playing a role in formulating the constraints (see below).

3.3 Constraints

To build a valid solution, a number of constraints have to be satisfied. For the validity of the function g, it is necessary that each user is assigned to a data processing path of the application that the user wants to use, i.e.:

$$\forall u \in U: \quad g(u) \in \mathcal{P}_{a_u}. \tag{2}$$

The following capacity constraint ensures that the total size of the modules that are placed on a node n does not exceed the capacity of n:

$$\forall n \in N: \quad \sum_{m \in f^{-1}(n)} S_m(w_m) \leq C_n. \tag{3}$$

To formulate the latency constraints, we first compute the latency perceived by a user u, depending on the f and g functions. For this purpose, let $P = (m_1, m_2, \ldots, m_{\kappa_P})$ be a data processing path, where $m_i \in M \cup \{*\}$ for all $1 \leq i \leq \kappa_P$, and let $g(u) = P$, i.e., user u is assigned to data processing path P. The latency between m_i and m_{i+1} (where $1 \leq i \leq \kappa_P - 1$), perceived by user u, is given by

$$\lambda_{P,i,u} = \begin{cases} \ell_{f(m_i),f(m_{i+1})} & \text{if } m_i, m_{i+1} \in M \\ \ell_{f(m_i),n_u} & \text{if } m_i \in M \text{ and } m_{i+1} = * \\ \ell_{n_u,f(m_{i+1})} & \text{if } m_i = * \text{ and } m_{i+1} \in M \\ 0 & \text{if } m_i = m_{i+1} = * \end{cases}$$

With this notation, the latency perceived by user u can be computed as

$$L(u) = \sum_{i=1}^{\kappa_P - 1} \lambda_{P,i,u},$$

and the latency constraint can be formulated as follows:

$$\forall u \in U: \quad L(u) \leq L_P, \text{ where } P = g(u). \tag{4}$$

Now we formulate the constraints stemming from security and privacy requirements. The module-level location constraints ensure that modules are not placed on illegal nodes:

$$\forall m \in M: \quad f(m) \notin I_m. \tag{5}$$

User-level location constraints enforce that modules processing data of a given user are not placed on the disallowed nodes:

$$\forall u \in U,\ \forall m \in M_{a_u}: \quad m \in g(u) \Rightarrow f(m) \notin I_{m,u}. \tag{6}$$

(Note that $m \in g(u)$ means that module m processes data of user u.)

Colocation constraints stipulate that given pairs of modules are not allowed to be placed on the same node:

$$\forall (m_1, m_2) \in \mathcal{I}: \quad f(m_1) \neq f(m_2). \tag{7}$$

k-anonymity constraints ensure that a sufficient number of users are assigned to modules that require this to achieve the predefined level of anonymity:

$$\forall m \in M: \quad w_m \geq k_m. \tag{8}$$

Thus, our aim is to find functions f and g such that constraints (2)–(8) are satisfied. It should be noted that while some constraints only relate to the application placement f (e.g., (7)) or only to the user assignment g (e.g., (2)), several constraints express the interdependence of f and g (e.g., (6)). This underlines the importance of jointly handling application placement and user assignment.

3.4 Discussion

We would like to emphasize that our problem formulation is an abstraction. Applying our formulation in practice will raise some questions. In particular, obtaining the input data (e.g., the set of disallowed nodes for a module) may be challenging and may require complex manual or automated processes (e.g., in the field of risk management), which are out of the scope of this paper. Also, in a specific system, possibly only a subset of the types of security and privacy requirements considered in this paper is relevant.

Another aspect is what happens if the problem defined here is not solvable. In this case, either the design of the system needs to be changed, or the requirements may have to be re-negotiated.

4 Algorithm using mixed integer programming

To solve the problem defined in Section 3, we devise an algorithm, which reads the inputs and transforms them to a quadratically constrained mixed integer

Table 2. Variables

Variable	Index set	Range
$x_{m,n}$	$m \in M, n \in N$	$\{0,1\}$
$y_{u,P}$	$u \in U, P \in \mathcal{P}$	$\{0,1\}$
w_m	$m \in M$	$\mathbb{N}$
$\lambda_{P,i,u}$	$P \in \mathcal{P}, i \in \{1,\ldots,\kappa_P - 1\}, u \in U$	$\mathbb{R}_{\geq 0}$

programming formulation. This mixed integer program is solved using an appropriate external solver. Finally, our algorithm transforms the output of the solver back to create the output of the problem as defined in Section 3.

To create the mixed integer program, we first define appropriate variables, which are summarized in Table 2. The function f is encoded using a set of binary variables. For a module $m \in M$ and a node $n \in N$:

$$x_{m,n} = \begin{cases} 1 & \text{if } f(m) = n \\ 0 & \text{otherwise} \end{cases}$$

The function g is encoded using another set of binary variables. For a user $u \in U$ and a data processing path $P \in \mathcal{P}$:

$$y_{u,P} = \begin{cases} 1 & \text{if } g(u) = P \\ 0 & \text{otherwise} \end{cases}$$

The following equation ensures that each module is placed on exactly one node:

$$\forall m \in M: \quad \sum_{n \in N} x_{m,n} = 1 \tag{9}$$

The following pair of equations ensure that each user is assigned to exactly one of the data processing paths of the application that the user wants to use, and to no data processing path of any other application:

$$\forall u \in U: \quad \sum_{P \in \mathcal{P}_{a_u}} y_{u,P} = 1 \tag{10}$$

$$\forall u \in U,\ \forall P \notin \mathcal{P}_{a_u}: \quad y_{u,P} = 0 \tag{11}$$

In addition, a set of integer variables is used to reflect the number of users served by each module, corresponding to the quantity w_m in Section 3. By a slight abuse of notation, we use w_m here as a variable: for a module $m \in M$, the number of users served by m is captured by variable w_m. The value of w_m is determined by the values of the y variables as follows:

$$w_m = \sum_{P \in \mathcal{P}(m)} \sum_{u \in U} y_{u,P} \tag{12}$$

To see the correctness of (12), it should be noted that the value of the inner sum is the number of users assigned to data processing path P. In addition, the equations (10)–(11) ensure that for each user u, $y_{u,P}$ will be 1 for exactly one data processing path, so that no user is counted twice in (12).

Using the w_m variables, the capacity constraint can be formulated as follows:

$$\forall n \in N: \quad \sum_{m \in M} x_{m,n} \cdot (\alpha_m + \beta_m \cdot w_m) \leq C_n \tag{13}$$

To calculate latencies, we consider a data processing path $P = (m_1, m_2, \ldots, m_{\kappa_P})$, where $m_i \in M \cup \{*\}$ for all $1 \leq i \leq \kappa_P$. Again by a slight abuse of notation, we use $\lambda_{P,i,u}$ to denote the real-valued variable that captures the latency between m_i and m_{i+1} for user u. The value of $\lambda_{P,i,u}$ can be computed from the x variables as follows:

$$\lambda_{P,i,u} = \begin{cases} \displaystyle\sum_{n \in N} \sum_{n' \in N} \ell_{n,n'} \cdot x_{m_i,n} \cdot x_{m_{i+1},n'} & \text{if } m_i, m_{i+1} \in M \\ \displaystyle\sum_{n \in N} \ell_{n,n_u} \cdot x_{m_i,n} & \text{if } m_i \in M,\ m_{i+1} = * \\ \displaystyle\sum_{n \in N} \ell_{n_u,n} \cdot x_{m_{i+1},n} & \text{if } m_i = *,\ m_{i+1} \in M \\ 0 & \text{if } m_i = m_{i+1} = * \end{cases} \tag{14}$$

To see the correctness of (14), it should be noted that, because of (9), exactly one term will be non-zero in each sum.

Using the $\lambda_{P,i,u}$ variables, the latency constraint can be formulated as follows:

$$\forall u \in U: \sum_{P \in \mathcal{P}_{a_u}} y_{u,P} \cdot \left(\sum_{i=1}^{\kappa_P - 1} \lambda_{P,i,u} \right) \leq \sum_{P \in \mathcal{P}_{a_u}} y_{u,P} \cdot L_P \tag{15}$$

To see the correctness of (15), it should be noted that, because of (10)–(11), $y_{u,P}$ will be non-zero for exactly one P.

Module-level location constraints can be easily formulated using the x variables as follows:

$$\forall m \in M,\ \forall n \in I_m: \quad x_{m,n} = 0. \tag{16}$$

User-level location constraints depend on both the x and y variables and can be formulated as follows:

$$\forall u \in U,\ \forall m \in M_{a_u},\ \forall n \in I_{m,u}: x_{m,n} \leq 1 - \sum_{P \in \mathcal{P}(m)} y_{u,P} \tag{17}$$

To see the correctness of (17), it should be noted that the sum on the right-hand side is 1 if module m processes data of user u and 0 otherwise. In the former case, (17) ensures that $x_{m,n} = 0$, while in the latter case, (17) imposes no constraint.

Colocation constraints can be formulated using the x variables as follows:

$$\forall (m_1, m_2) \in \mathcal{I},\ \forall n \in N: \quad x_{m_1,n} + x_{m_2,n} \leq 1. \tag{18}$$

Table 3. Settings used in the experiments

Setting	Values
Size	Each module: $30 + 5 \cdot w_m$
Capacity	Smart Home Controllers: 100
	Gateway: 200
	Cloud: ∞
Latency	Users – Smart Home Controllers: 0
	Smart Home Controllers – Gateway: 10
	Gateway – Cloud: 100
	Maximum allowed latency for the data paths: 300

Finally, k-anonymity constraints can be directly formulated using the w variables as follows:

$$\forall m \in M: \quad w_m \geq k_m. \tag{19}$$

As can be easily seen, Equations (9)–(19) describe exactly the constraints that a solution to the problem of Section 3 has to satisfy. Hence, the mixed integer program is solvable if and only if the original problem was solvable. It should also be noted that the constraints are either linear (e.g., (9)), or quadratic (e.g., (15)) in the variables.

A solution to the mixed integer program can be transferred back to a solution to the problem of Section 3 by using the following rules:

- For a module $m \in M$, $f(m)$ is the single node $n \in N$ for which $x_{m,n} = 1$.
- For a user $u \in U$, $g(u)$ is the single data processing path $P \in \mathcal{P}$ for which $y_{u,P} = 1$.

5 Evaluation

We implemented our approach in the form of a Java program, which uses the Gurobi Optimizer[1] version 9.0.1 to solve mixed integer programs. To foster reproducibility, we made our implementation available online[2].

5.1 Example application

To validate the applicability of our approach, we first apply it to the example of Section 2. The values that are used for the parameters of the infrastructure and the application are given in Table 3.

The result of applying our approach is shown in Fig. 2. The arrows between users and data processing paths show the g function (i.e., the assignment of users) computed by our approach. The arrows between data processing paths and modules show which modules each data processing path consists of, which

[1] https://www.gurobi.com
[2] https://sourceforge.net/p/vm-alloc/sec-place-usr-asgn

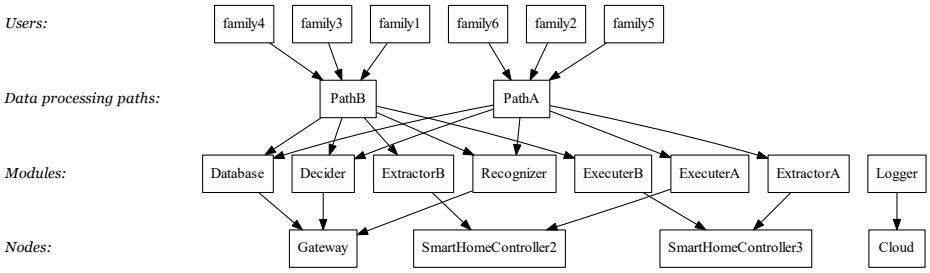


Fig. 2. Result of applying the proposed algorithm to the example of Section 2

is given as part of the input. The arrows between modules and nodes show the f function (i.e., the placement of application modules) computed by our approach.

It can be verified that all constraints described in Section 2 are satisfied in the computed solution. In particular, the Database is not in the Cloud, modules processing data of Family 2 are not placed on Smart Home Controller 1, Executer A and Executer B are on different nodes, and Extractor A and B are assigned at least three users, as stipulated by the security and privacy requirements.

The experiment also shows that finding an application placement and user assignment satisfying all constraints is quite complicated and challenging even for problem instances of moderate size. Manually solving the problem is hard and may take a long time. Our algorithm, however, solves the problem in a fraction of a second.

5.2 Scalability

In the next set of experiments, we investigated the scalability of the proposed approach. This is important because our method is based on quadratically constrained mixed integer programming, which has exponential worst-case complexity. Hence it is interesting to evaluate how big problem instances can be solved in acceptable time.

To investigate the effects of increasing problem instance size, we scale the Smart Bell system of Section 2 to an increasing number of homes. We simultaneously increase the number of nodes (adding new Smart Home Controller nodes), the number of modules (adding new Extractor and Executer instances) and corresponding data processing paths, as well as the number of users (one more family with each new home). As a result, also the number of constraints stemming from security and privacy requirements increases. Table 4 summarizes the parameters of the considered problem instances.

Fig. 3 shows the execution time of our algorithm for increasing problem size. The execution time includes the time to create the mixed integer program, solve it with the external solver, and retrieve the application placement and user assignment from the results. The experiments were carried out on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM. Note that, although the horizontal axis in Fig. 3 only shows the number of users

Table 4. Scaling experiment. S&P: security & privacy

Nodes	Modules	Users	S&P constraints	Time [s]
11	8	6	12	0.16
20	12	12	19	0.75
29	16	18	26	1.43
38	20	24	33	3.99
47	24	30	40	8.76
56	28	36	47	16.25
65	32	42	54	28.86
74	36	48	61	45.56
83	40	54	68	74.38
92	44	60	75	109.61

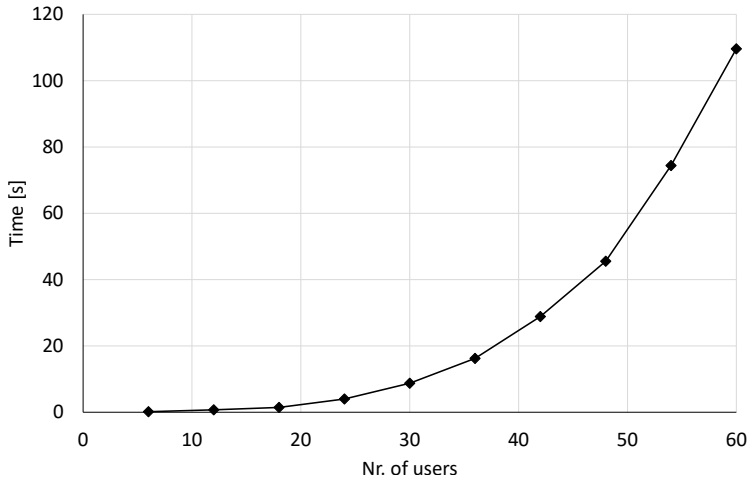


Fig. 3. Scaling behavior of the algorithm's execution time

(which equals the number of smart homes considered), the problem instances also grow at the same time in the other dimensions as shown in Table 4.

From Fig. 3 it is clear that the execution time indeed exhibits rapid growth. Nevertheless, even the largest problem instance, which comprises the joint optimization of module placement and user assignment for 60 smart homes, takes less than 2 minutes. Thus we can conclude that our approach can solve even quite complex problem instances (which would be an overwhelming challenge for humans) with acceptable execution time, using a commodity computer.

5.3 Impact of security and privacy constraints

We also investigate how an increasing number of security and privacy constraints impacts the solvability of the problem instances and the execution time of our algorithm.

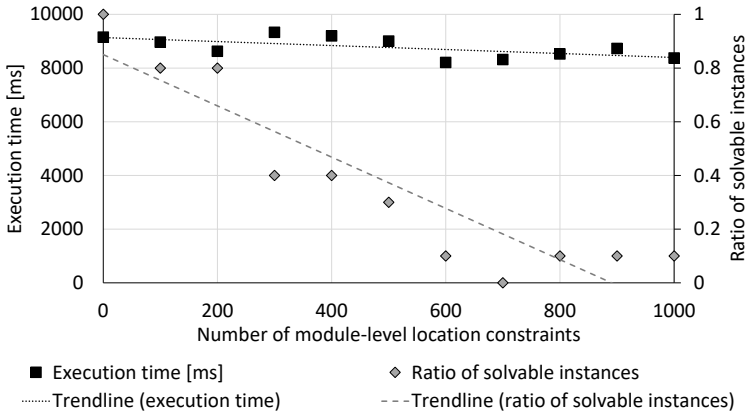


Fig. 4. Impact of the number of *module-level location constraints*

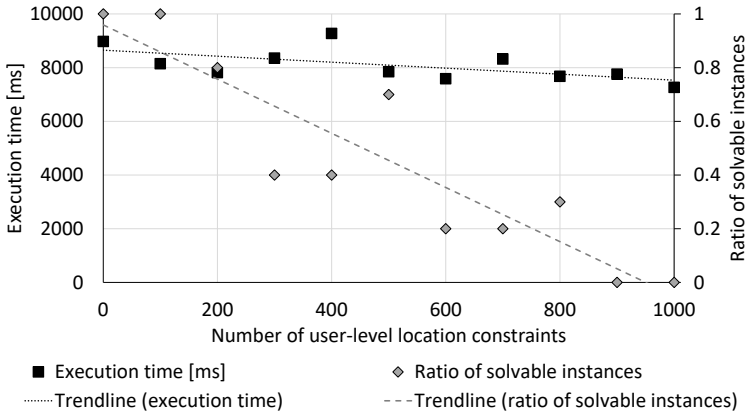


Fig. 5. Impact of the number of *user-level location constraints*

Figures 4-6 show the impact of location and colocation constraints. In these experiments, we started from the instance of the previous scaling experiment (see Table 4) with 30 homes, which includes 47 nodes, 24 modules, and 30 users. Unlike in the previous scaling experiment, we now started with an empty set of security and privacy constraints, and then added an increasing number of a specific type of constraint. Fig. 4 shows the impact of adding up to 1000 module-level location constraints for randomly chosen modules and nodes. Fig. 5 shows the impact of adding up to 1000 user-level location constraints for randomly chosen users, modules, and nodes. Fig. 6 shows the impact of adding up to 100 colocation constraints for randomly chosen pairs of modules. In each of these cases, the data points represent the average of 10 measurements each. Each figure shows the impact of the number of constraints (horizontal axis) on the execution time of the algorithm (black squares, left vertical axis) and on the ratio

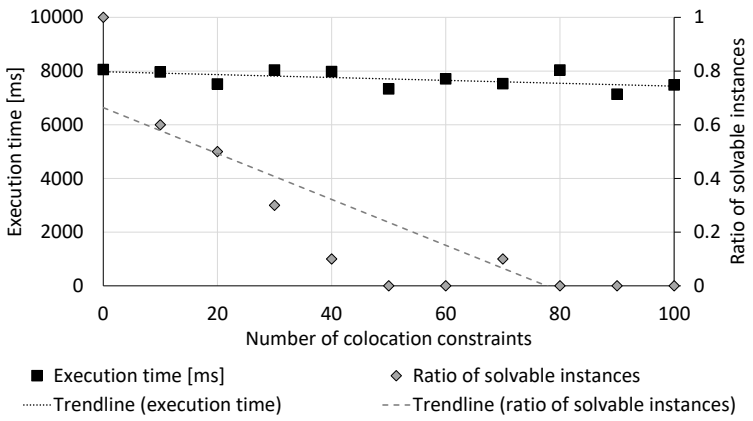
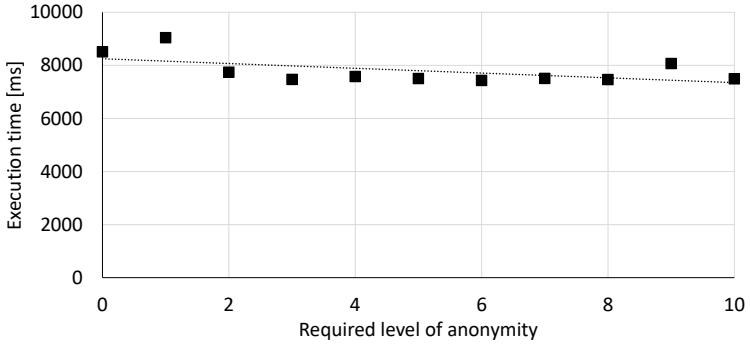


Fig. 6. Impact of the number of *colocation constraints*

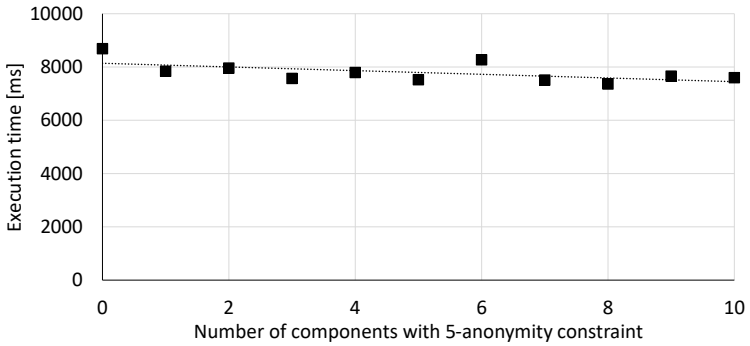
of solvable problem instances (gray diamonds, right vertical axis). Furthermore, the trendlines for the execution time (black dotted line) and for the ratio of solvable problem instances (gray dashed line) are also shown.

In each figure, the same pattern can be observed. When the number of constraints is low, most problem instances are solvable. As the number of constraints increases, the ratio of solvable problem instances decreases. With the highest number of constraints, most problem instances are not solvable anymore. Hence, security and privacy constraints have a large impact on whether a solution that satisfies all constraints can be found. On the other hand, the impact of security and privacy constraints on the algorithm's execution time is very limited. The algorithm's execution time seems to have a small peak roughly at the point where the ratio of solvable instances drops. This is in line with previous experience on other combinatorial problems: when there are few constraints, it is relatively easy to find a solution and when there are many constraints, it is relatively easy to come to a contradiction, but deciding solvability in-between is more difficult [16]. Overall, there is a slight negative correlation between the number of constraints and the algorithm's execution time, as demonstrated by the negative slope of the trendlines. This may be attributed to the fact that a higher number of constraints enables the solver to more effectively prune the parts of the search space that certainly do not contain any solutions.

Regarding the k-anonymity constraints, two different parameters can be varied, as shown in Fig. 7. We start from the same setup as in the previous experiment, i.e., with 30 homes, which includes 47 nodes, 24 modules, 30 users, and no security or privacy constraints. In the experiment whose result is shown in Fig. 7(a), all 10 Extractor components are associated with a k-anonymity constraint, and we vary k from 0 to 10. The figure shows the impact on the algorithm's execution time. For $k \leq 3$, the problem is solvable, for $k \geq 4$, the problem is not solvable. In the experiment whose result is shown in Fig. 7(b), we apply 5-anonymity constraints to a varying number of the 10 Extractor com-



(a) Impact of the required level of anonymity



(b) Impact of the number of components with 5-anonymity constraints

Fig. 7. Impact of the k-anonymity constraints

ponents. The figure shows the impact on the algorithm's execution time. For at most 6 components with 5-anonymity constraints, the problem is solvable, for 7 or more components with 5-anonymity constraints, the problem is not solvable. Altogether, Fig. 7 supports the same conclusions that we could draw from Figures 4-6: the number and tightness of security and privacy constraints has only a minor impact on the algorithm's execution time, and more or tighter constraints tend to slightly decrease the algorithm's execution time.

5.4 Summary

Summarizing the experience from all experiments, we can state that our algorithm can effectively solve the joint problem of application placement and user assignment, even for problem instances that would be very hard to solve for a human. With growing problem instances, also the algorithm's execution time grows quickly, but the algorithm is still able to solve quite large problem instances in acceptable time. Security and privacy constraints significantly influence the problem's solvability, but only slightly influence the algorithm's execution time.

6 Related work

In the following, we discuss related work structured according to whether it applies to application placement or user assignment.

Application placement. Optimizing the placement of application modules in heterogeneous infrastructures has been the subject of intensive research [6, 17]. However, most of the existing approaches either completely ignore security and privacy requirements or handle them in a rudimentary way.

A simple way of representing security requirements of application components and security capabilities of infrastructure nodes is by using security levels. Goettelmann et al. used this approach for specifying security constraints, and then applied a combination of a greedy algorithm and tabu search for optimizing the placement [14]. Wen et al. also used a similar security model and a custom heuristic algorithm for placement optimization [26, 27]. Mezni et al. also adopted a similar security model and used particle swarm optimization to find a good placement [20]. In contrast to these approaches, we use a rich set of rigorous security and privacy constraints. In addition, we apply an exact algorithm that guarantees the fulfillment of all stipulated constraints.

Instead of security levels, a more precise way of capturing security constraints is by defining the specific security controls required by the different application modules, respectively offered by the different infrastructure nodes. Massonet et al. used this approach for specifying security constraints [19]. They proposed a method based on constraint programming that finds an optimized placement respecting the given security requirements. Forti et al. also used a similar approach, extended with probabilities and trust relations among stakeholders [13]. Our approach also allows to capture security constraints stemming from the security controls required by application modules and offered by infrastructure nodes, in the form of location constraints, but also many other types of constraints – including co-location and k-anonymity constraints – not supported by the mentioned previous works.

Co-location constraints have been taken into account by some previous approaches. Fdhila et al. considered such constraints when partitioning and placing composite applications on federated clouds [12]. Agarwal and Duong also focused on the risks of co-location in public infrastructure clouds [1]. In our earlier work, we devised custom heuristics for handling co-location constraints during application placement, also taking into account the availability of secure hardware enclaves [18]. The approach of the present paper also takes into account co-location constraints, but in combination with several other types of security and privacy requirements.

Workflow scheduling was considered in conjunction with data protection concerns by Wen et al. [25]. However, in that work, data protection constraints are limited to the specification of the allowed set of data centers for a task. Also for the placement of applications on a fog infrastructure, several approaches take security and privacy concerns into account by means of constraining the placement of certain application modules to trusted hosts [28, 21, 7]. While our approach

also supports such location constraints, it can take into account various other types of security and privacy constraints as well.

Yuchi and Shetty define simple metrics to quantify the vulnerability of virtual machines containing application modules and the survivability of physical infrastructure nodes [30]. They use these pieces of information to place the application modules on the nodes with the aim of minimizing the overall risks. They propose a heuristic algorithm for this purpose. In contrast, our approach can guarantee the fulfillment of strict security and privacy requirements.

User assignment. The assignment of users to different modules has also been investigated in different contexts.

Deng et al. address the problem of assigning requests of users to fog devices and cloud servers, with the goal of balancing latency and energy consumption objectives [10]. A similar problem is addressed by Shah-Mansouri and Wong, aiming to allocate fog and cloud resources to users with the objective of serving as many users as possible with as low latency as possible [23]. Xiao and Krunz also consider the assignment of users' workload to fog nodes, with the aim of improving quality of experience for the users [29]. Chen et al. address the allocation of user requests to a heterogeneous set of network resources in a mobile-edge cloud computing scenario, with the objective of minimizing the overhead observed by users [9]. Dräxler et al. assign users to instances of network services, with the objective of minimizing the number of violations of capacity constraints [11].

Our approach also takes into account the key objectives of these works, including low latency and good utilization of the available capacity of the nodes. On the other hand, none of the above works consider security and privacy requirements, despite the huge importance of such requirements in modern computing systems. Our approach significantly advances the state of the art by adding security and privacy constraints.

7 Conclusions and future work

In this paper, we have addressed the joint problem of application module placement and user assignment in the context of heterogeneous applications, heterogeneous infrastructure, and different types of security and privacy constraints, in addition to the more traditional constraints on capacity and latency. Beside formally defining the problem, we devised an algorithm to solve the problem by means of quadratically constrained mixed integer programming. The algorithm is guaranteed to find a placement of the application modules and assignment of the users satisfying all constraints, whenever this is possible.

We demonstrated the applicability of the proposed approach by applying it to an IoT system from the smart home domain. In addition, we assessed the scalability of the algorithm by applying it to problem instances of increasing size. We also investigated the impact of the number of security and privacy constraints. The experiments showed that the proposed algorithm can solve even quite large problem instances in acceptable time using a commodity computer.

Several interesting paths for future research can be identified. One promising direction is to consider an online variant of the module placement and user assignment problem, in which the aim is to react to changes (e.g., the appearance of new users) by adapting the module placement and/or the user assignment so that the continued satisfaction of the requirements is guaranteed. Another interesting possibility is the parallelization of the proposed algorithm using multiple nodes, so as to reduce the execution time of the algorithm. (For distributed solving of integer programs, see [8] and references therein.)

Acknowledgments. This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant 871525 (FogProtect).

References

1. Agarwal, A., Duong, T.N.B.: Secure virtual machine placement in cloud data centers. Future Generation Computer Systems **100**, 210–222 (2019)
2. Alrawais, A., Alhothaily, A., Hu, C., Cheng, X.: Fog computing for the Internet of Things: Security and privacy issues. IEEE Internet Computing **21**(2), 34–42 (2017)
3. Ayed, D., Jaho, E., Lachner, C., Mann, Z.Á., Seidl, R., Surridge, M.: FogProtect: Protecting sensitive data in the computing continuum. In: Advances in Service-Oriented and Cloud Computing – International Workshops of ESOCC 2020. pp. 179–184 (2021)
4. Baresi, L., Mendonça, D.F., Garriga, M., Guinea, S., Quattrocchi, G.: A unified model for the mobile–edge–cloud continuum. ACM Transactions on Internet Technology **19**(2), art. 29 (2019)
5. Bellendorf, J., Mann, Z.Á.: Classification of optimization problems in fog computing. Future Generation Computer Systems **107**, 158–176 (2020)
6. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog: State of the art and open challenges. Software: Practice and Experience **50**(5), 719–740 (2020)
7. Brogi, A., Forti, S., Ibrahim, A.: Predictive analysis to support fog application deployment. Fog and edge computing: Principles and paradigms pp. 191–222 (2019)
8. Chamanbaz, M., Notarstefano, G., Sasso, F., Bouffanais, R.: Randomized constraints consensus for distributed robust mixed-integer programming. IEEE Transactions on Control of Network Systems **8**(1), 295–306 (2021)
9. Chen, X., Jiao, L., Li, W., Fu, X.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Transactions on Networking **24**(5), 2795–2808 (2015)
10. Deng, R., Lu, R., Lai, C., Luan, T.H., Liang, H.: Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet of Things Journal **3**(6), 1171–1181 (2016)
11. Dräxler, S., Karl, H., Mann, Z.Á.: JASPER: Joint optimization of scaling, placement, and routing of virtual network services. IEEE Transactions on Network and Service Management **15**(3), 946–960 (2018)
12. Fdhila, W., Dumas, M., Godart, C., García-Bañuelos, L.: Heuristics for composite web service decentralization. Software & Systems Modeling **13**(2), 599–619 (2014)
13. Forti, S., Ferrari, G.L., Brogi, A.: Secure cloud-edge deployments, with trust. Future Generation Computer Systems **102**, 775–788 (2020)

14. Goettelmann, E., Fdhila, W., Godart, C.: Partitioning and cloud deployment of composite web services under security constraints. In: IEEE International Conference on Cloud Engineering (IC2E). pp. 193–200. IEEE (2013)
15. Jiang, W., Clifton, C.: Privacy-preserving distributed k-anonymity. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 166–177. Springer (2005)
16. Mann, Z.Á.: Optimization in computer engineering–Theory and applications. Scientific Research Publishing, Inc. USA (2011)
17. Mann, Z.Á.: Secure software placement and configuration. Future Generation Computer Systems **110**, 243–253 (2020)
18. Mann, Z.Á., Metzger, A.: Optimized cloud deployment of multi-tenant software considering data protection concerns. In: 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). pp. 609–618. IEEE (2017)
19. Massonet, P., Luna, J., Pannetrat, A., Trapero, R.: Idea: Optimising multi-cloud deployments with security controls as constraints. In: International Symposium on Engineering Secure Software and Systems. pp. 102–110. Springer (2015)
20. Mezni, H., Sellami, M., Kouki, J.: Security-aware SaaS placement using swarm intelligence. Journal of Software: Evolution and Process **30**(8), e1932 (2018)
21. Nardelli, M., Cardellini, V., Grassi, V., Presti, F.L.: Efficient operator placement for distributed data stream processing applications. IEEE Transactions on Parallel and Distributed Systems **30**(8), 1753–1767 (2019)
22. Roman, R., Lopez, J., Mambo, M.: Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems **78**, 680–698 (2018)
23. Shah-Mansouri, H., Wong, V.W.: Hierarchical fog-cloud computing for IoT systems: A computation offloading game. IEEE Internet of Things Journal **5**(4), 3246–3257 (2018)
24. Sweeney, L.: k-anonymity: A model for protecting privacy. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **10**(05), 557–570 (2002)
25. Wen, Y., Liu, J., Dou, W., Xu, X., Cao, B., Chen, J.: Scheduling workflows with privacy protection constraints for big data applications on cloud. Future Generation Computer Systems **108**, 1084–1091 (2020)
26. Wen, Z., Cala, J., Watson, P.: A scalable method for partitioning workflows with security requirements over federated clouds. In: IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom). pp. 122–129 (2014)
27. Wen, Z., Cala, J., Watson, P., Romanovsky, A.: Cost effective, reliable and secure workflow deployment over federated clouds. IEEE Transactions on Services Computing **10**(6), 929–941 (2017)
28. Xia, Y., Etchevers, X., Letondeur, L., Coupaye, T., Desprez, F.: Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog. In: Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC). pp. 751–760 (2018)
29. Xiao, Y., Krunz, M.: QoE and power efficiency tradeoff for fog computing networks with fog node cooperation. In: INFOCOM – IEEE Conference on Computer Communications (2017)
30. Yuchi, X., Shetty, S.: Enabling security-aware virtual machine placement in IaaS clouds. In: IEEE Military Communications Conference (MILCOM). pp. 1554–1559 (2015)