Computer Networks 185 (2021) 107727

Contents lists available at ScienceDirect

Computer Networks

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

SYNCOP: An evolutionary multi-objective placement of SDN controllers for optimizing cost and network performance in WSNs

Shirin Tahmasebi [a], Nayereh Rasouli [b], Amir Hosein Kashefi [c], Elmira Rezabeyk [d], Hamid Reza Faragardi [e,*]

[a] Sharif University of Technology, Tehran, Iran
[b] Hashtgerd Branch, Islamic Azad University, Hashtgerd, Iran
[c] South Tehran Branch, Islamic Azad University, Tehran, Iran
[d] Amirkabir University of Technology, Tehran, Iran
[e] KTH Royal Institute of Technology, Stockholm, Sweden

ARTICLE INFO

Keywords:
Wireless sensor networks
Software defined networks
Controller node placement
Cuckoo optimization algorithm
Synchronization cost

ABSTRACT

Due to the highly dynamic nature of Wireless Sensor Networks (WSN), Software-Defined Network (SDN) is a promising technology to ease network management by providing a logically centralized control plane. It is a common approach to employ multiple SDN controllers to form a physically distributed SDN to achieve better fault tolerance, boost scalability, and enhance performance. Despite physical distribution, since the notion behind SDN is to logically centralize network management, it is essential to provide a consistent view of the network's state for all controllers. Deploying multiple controllers result in higher synchronization and deployment cost. Since network performance and inter-controller synchronization cost seem to be contradicting objectives, it is a research challenge to choose the best placement of SDN controllers to optimize both the performance and synchronization cost of an SDN-enabled WSN simultaneously.

In this paper, we first formulate the controller placement problem as a multi-objective optimization problem. In this regard, multiple constraints are considered, including reliability, fault tolerance, performance in terms of latency, synchronization overhead, and deployment cost. Moreover, we leverage the Cuckoo optimization algorithm, a nature-inspired population-based meta-heuristic algorithm to solve the optimization problem. This algorithm seeks to find the global optimum by imitating brood parasitism of some cuckoo species. Finally, to evaluate our proposed method, we compare it against several existing methods in the literature. The experiments reveal that our proposed method considerably outperforms existing methods, namely Simulated Annealing (SA) and Quantum Annealing (QA), in terms of both performance and synchronization cost. Additionally, our proposed algorithm, in contrast to Integer Linear Programming (ILP), is considerably more scalable, which makes it applicable for large-scale WSNs.

1. Introduction

Recently, using SDN in Wireless Sensor Networks (WSNs) has become an increasingly important trend. Due to the high dynamicity of WSNs, using SDN in WSNs can provide several benefits, including improving flexibility, boosting scalability, eliminating the complexity of the network, better configuration and management, and replacing rigidity to policy changes [1,2]. Dynamic reconfiguration of the network is a useful feature, especially in harsh environments, where wireless links could be highly unreliable, and thus network routing should be updated frequently [1,3]. Moreover, large scale WSNs will require multiple SDN controllers in order to manage their configuration.

SDN controllers are considered to be resource-intensive devices which are capable of storing a high volume of data and doing complex computational operations. However, WSN devices are faced with severe limitations in terms of processing power, storage, and battery resources and are not capable of performing resource-intensive operations. Hence, WSN devices are not resourceful enough to implement SDN controller functionalities. A promising solution to implement the SDN controller is to integrate additional node(s) within WSNs, which are responsible for executing the SDN controller functionalities. Although the notion behind the control plane in software-defined networks is based on a centralized fashion, the SDN controller

* Corresponding author.
E-mail addresses: shtahmasebi@ce.sharif.edu (S. Tahmasebi), n.rasouli@hiau.ac.ir (N. Rasouli), a.kashefi@azad.ac.ir (A.H. Kashefi), elmira.rezabeyk@aut.ac.ir (E. Rezabeyk), hrfa@kth.se (H.R. Faragardi).

https://doi.org/10.1016/j.comnet.2020.107727
Received 6 August 2020; Received in revised form 20 November 2020; Accepted 30 November 2020
Available online 11 December 2020

must be physically distributed among multiple nodes to achieve higher performance, scalability, and fault tolerance [4,5].

If we adopt a distributed version of the SDN controller in WSNs, then the question is how to deploy the SDN controller nodes in the network. How many nodes are required, and where should they be placed? Two important Quality of Service (QoS) elements should be taken into account when it comes to the deployment of multiple controller nodes in a WSN, namely, (i) reliability and (ii) latency.

Reliability is a challenging issue in WSNs since low-power links are highly unreliable, and the link quality fluctuates a lot [6]. Link failure can cause node inaccessibility to the network, disconnections, network performance degradation, and eventually node failure. In SDN-enabled WSNs, it is highly important to guarantee wireless connection to SDN controllers as they are acting as the brain of the network, and provide network reconfiguration. Thus, it is crucial to provide network reliability by replicating SDN controllers in order to avoid a single point of failure [7]. Hence, in the network architecture, multiple controllers should be accommodated to achieve higher reliability.

Latency is an important performance metric that can be affected considerably by the placement of controller nodes. A tactful placement of controller nodes can improve the network performance by reducing the distance between sensor nodes and the controller(s) covering the sensors. A shorter distance between controllers and sensors also decreases the network traffic and saves energy consumption. Thus, one of the major parameters for placing controllers in a sensing area is the distance (in terms of the number of hops) to sensor nodes.

The physically distributed control plane in SDN-enabled WSNs could result in more reliability, fault tolerance, timeliness, and better performance. However, it raises two other issues. Firstly, to make the control plane logically centralized, it is necessary to synchronize controllers and provide a consistent view of the network's state for all of them. Secondly, using multiple controllers would result in an increase in deployment cost. Thus, a physically distributed control plane can provide such benefits at the expense of incurring more synchronization and deployment cost. In [4], the trade-off between the synchronization cost of multiple controllers and network latency was discussed. The results indicated the feasibility of multi-controller deployments in terms of network latency if the right number of controllers are deployed.

In this paper, we aim to resolve the shortcomings of our previous work, [8], in order to propose a more efficient approach for controller placement in the Industrial Internet of Things (IIoT). In this regard, we intend to find an optimal controller placement to maximize the network performance subject to (i) a specified upper bound for the synchronization cost that limits the maximum number of controllers, and (ii) reliability constraints. Thus, we state a multi-objective problem in which synchronization cost and reliability are considered as two contradicting objectives. We formulate the problem as an optimization problem, which is then modeled as an Integer Linear Programming (ILP). The proposed ILP model is solved using the *Cuckoo SYNchronization-aware COntroller Placement (SYNCOP)* algorithm, and the results are compared not only with the recently proposed methods in the literature but also with the CPLEX ILP Solver.

Contributions. In this paper, an evolutionary solution for placement of SDN controllers problem is introduced to improve performance in WSNs subject to a certain budget limiting the maximum number of controllers, and the reliability constraint. Hence, our major contributions are listed as follows:

1. Targeting the optimal controller placement in SDN-enabled WSNs.
2. Considering several metrics in formulating our optimization problem, namely: reliability, performance (in terms of the number of hops between controllers and sensor nodes), inter-controller synchronization cost, and deployment cost.
3. Proposing the SYNCOP algorithm to tackle the placement optimization problem. SYNCOP considerably outperforms the existing (meta)heuristic algorithms in the literature, including Quantum Annealing (QA). Moreover, in contrast to ILP, it is a scalable algorithm and can be applied appropriately in large-scale WSNs.

Organization of the Paper. In Section 3, a comprehensive review of related work is presented. In Section 4, we describe the problem and our assumptions, which are then followed by formulating an optimization problem. Section 5 presents the algorithm proposed to discuss the optimization problem. In Section 6, the performance of the proposed method is investigated. Finally, in Section 7, we finalize our paper by providing a summary along with future directions.

2. Background

The origin of the Software-Defined Network (SDN) refers back to a research collaboration between Stanford University and the University of California at Berkeley [9]. The main idea behind SDN is to separate data and control planes in order to centralize network management. Using SDN for centralizing network management has several benefits, such as network reconfiguration through on-the-fly programming [10]. SDN provides fine-grained information to select the best forwarder to form global resource optimization as compared to ad-hoc networks where the next-hop to forward data towards the destination is picked out by each node separately [9].

It is worth mentioning that due to different characteristics between WSNs and traditional wired networks, applying SDN in WSNs introduces several new challenges, including [11]:

- In a WSN, most of the devices face severe limitations in terms of processing power, storage, and battery resources. Hence, it is of paramount importance to reduce unnecessary packet transmission in SDN-enabled WSNs in order to reduce packet collision and extend network lifetime. However, in a wired network, the main goal is to minimize the response time.
- Unlike wired networks, the links in a WSN are highly unstable and unreliable. Therefore, WSN sensors are subject to several types of failures, namely, communication failures due to environmental obstacles and power failures due to constrained and insufficient battery resources. Thus, to overcome link unreliability, multipath solutions may be applied to increase the chance of successful data transmission that, in turn, will increase network overhead.

Accordingly, it can be concluded that the nature of WSNs is significantly more dynamic than wired networks. Hence, the controller placement in WSNs, in comparison to wired networks, is more critical and challenging and has a more profound impact on the performance, lifetime, and QoS of the network.

The efficient node placement in SDN-enabled WSNs has essential benefits in several application domains. For example, it can be used in industrial applications, such as smart factories, to enable environmental monitoring, enhance productivity, increase flexibility, improve energy consumption, and reduce maintenance costs. In such harsh environments, where performance and time sensitivity are of paramount importance, using a single controller to manage the whole network is not sufficient. Instead, it is more effective and also challenging to deploy multiple SDN controllers to ensure the satisfaction of requirements. [12–15]

3. Related work

Several works have addressed node placement in WSNs and IoT systems [16–18]. These studies concentrate on different types of nodes and different performance metrics in their node placement strategies. An overview of studies focusing on sink node and controller placement problems are provided in Section 3.1 and Section 3.2, respectively.

Furthermore, the physically distributed control plane in SDN networks would result in numerous benefits such as more scalability, fault tolerance, and better performance. However, to make the control plane logically centralized, it is necessary to synchronize the controllers and provide a consistent view of the network's state for all of them. Thus, a physically distributed control plane provides some benefits at the expense of synchronization cost. Inter-controller synchronization cost has also been explored in several prior studies, some focusing on wired SDNs, and others on wireless SDNs. These studies are explored in Sections 3.3 and 3.4.

3.1. Sink placement in WSNs

In [19], two algorithms were presented for sink placement in WSNs, called as GreedyMSP and GRASP-MSP, and two algorithms were presented for sink and relay placement in WSNs, named as Greedy-MSRP and GRASP-MSRP. Greedy-MSP and GRASP-MSP were designed to minimize the deployment cost, while ensuring double-coverage of each sensor node in the network. In [20], a Genetic Algorithm (GA) is proposed to deploy multiple sinks in order to minimize the worst-case delay in WSNs, while in [21], a Particle Swarm Optimization (PSO) is employed to calculate the number of sinks and their locations in order to reduce the path length between sensors and sinks. In the same context, in [22], a PSO-based algorithm is proposed to prolong the network lifetime by considering the Euclidean distance and hop count. Moreover, the authors focused on reducing energy consumption in WSNs in a random deployment of a grid topology network. In [23], an approximation algorithm (GREEDY-k-SPP) is proposed to solve the placement of multiple sinks problem, wherein the objective is to minimize the worst-case delay. Authors in [24] evaluated the effects of static and dynamic sink node placement based on various QoS metrics, namely packet delivery ratio, average throughput, and end-to-end delay. According to their experimental result, using a dynamic sink placement approach can lead to an improvement in all of these three QoS metrics, in comparison to the static sink placement approach.

3.2. Controller placement

Feixiang Li et al. in [25], addressed the controller placement problem in wired networks and sought to minimize the latency between nodes and controllers. Additionally, they proposed three heuristics to solve their ILP formulated problem: Cuckoo Search Algorithm (CSA), Genetic Algorithm (GA), and Particle Swarm Optimization (PSO). Due to the experimental results, although employing CSA results in longer processing time in comparison to the other two algorithms, it leads to the best performance among two others in terms of average latency between nodes and controllers.

In wireless networks, authors in [26] proposed a deployment strategy for multiple controller nodes, which is used to minimize the transmission delay of the network with satisfying copious critical requirements, namely deployment cost and reliability. The controller placement problem in wireless SDNs is also explored in [27] by focusing on different metrics such as link failure probability, average throughput in the southbound interface, latency, and transparency–which is considered as the latency caused by interference from controllers. Furthermore, although a common approach to measure latency is to use the Euclidean distance, in [27], a much more complicated model is introduced to reflect the effects of collision and interference on latency. Finally, they exploited a combination of hill-climbing algorithm and simulated annealing algorithm to present an efficient solution. In [28], the authors aimed to solve the controller placement problem considering delay and reliability. Moreover, they have investigated two different aspects of the controller placement problem; first, the way to select the optimum position of the controllers; and second, the way to choose the best set of switches to allocate to each controller. Hence, two meta-heuristic algorithms are used to address each of these two aspects: (i) Louvain algorithm to tackle the first aspect and (ii) PSO for the second one. Authors, in [29], considered network resilience and delay performance in the controller placement problem. They proposed a novel node metric called the nodal disjoint path (NDP). NDP is used to measure the importance of a node in terms of its path diversity to other nodes. Moreover, two greedy algorithms were leveraged to address the problem: (i) NDP-global and (ii) NDP-cluster. Since using a centralized control plane may result in a bottleneck, in [30], the authors presented a two-level hierarchy control framework that leads to a decentralized infrastructure. They named this hierarchy as a master–slave model and sought to optimize the controller placement in the slave layer. Moreover, they considered control delay and the control cost of critical nodes as the optimization objectives. Node criticality is considered as the importance of each node in terms of several factors. It is evaluated using Fuzzy AHP method based on several criteria such as device attributes (type, location, and owner), service attributes (delay requirements, privacy requirements, the service user and service profit), and control frequency (number of communications with the controller at one given time). At last, the optimization problem is addressed using Particle Swarm Optimization (PSO) algorithm.

3.3. Inter-controller synchronization overhead in wired SDN

In [4], two metrics are considered to calculate the inter-controller synchronization cost; the first metric is the synchronization delay, which is defined as the time it takes for a controller to be aware of a generated event from another controller; and the second is the amount of synchronization data that is exchanged between controllers. Their experimental results illustrated that the synchronization delay is negligible in most cases, and to gain more scalability and fault tolerance, it is reasonable to compromise on inter-controller synchronization cost. In [31], a framework is introduced for dynamic controller deployment and provisioning. The authors considered several factors for measuring the deployment cost such as flow setup, inter-controller, statistics collection, and reassignment costs. They formulated the problem as an ILP and proposed two heuristics algorithms: a greedy approach based on the knapsack problem (DCP-GK) and a meta-heuristic approach based on simulated annealing algorithm (DCP-SA). In [32], the controller placement problem is addressed with regard to several factors, namely: inter-controller synchronization cost, controller-switch communication cost, flow statistics collection cost, and measurement overhead. Finally, Discrete Approximation Algorithm (DAA) and Connectivity Ranking Algorithm (CRA) are used to approximate the solution. According to their results, leveraging the proposed approach could lead to 40% reduction in measurement overhead on average. The authors in [33] solved the controller placement problem by focusing on the minimization of the total required bandwidth for the inter-controller synchronization traffic and also represented experimental results in a realistic environment that was offered by an SDN testbed. In [34], the Pareto-based Optimal COntroller placement algorithm (POCO) is proposed to solve the controller placement problem in SDN-based core networks. Moreover, the authors considered the trade-off between several metrics, including different types of resilience and failure tolerance. Several metrics are taken into account to measure resilience and failure tolerance, such as node-controller latency, load balancing, and inter-controller synchronization latency. Finally, they presented a framework for resilience that prepared the operator of a network with all pareto-optimal placements.

3.4. Inter-controller synchronization overhead in wireless SDN

In [35], the authors focused on the inter-controller synchronization problem in wireless SDN networks. They considered both leaderless and leader-based strategies for controller synchronization. Their optimization objective was to reduce delay and controller synchronization overhead in both strategies. In [36], the controller-switch association

problem is addressed using a heuristic algorithm based on a combination of Dijkstra and K-means algorithms. They also solved the controller capacity matching problem by applying the Kuhn–Munkres (K-M) algorithm. Moreover, they sought to minimize the control plane delay by considering not only the delay between controllers and switches but also the inter-controller delay.

3.5. Comparison of different node-placement methods and strategies

In this subsection, we review the related works on the SDN node placement which have somehow considered inter-controller synchronization cost, whether in terms of synchronization overhead or in terms of inter-controller delay. In this regard, a comprehensive analysis of related work is summarized in Table 1. Each row of this table represents the details of a research paper—details such as which of our objectives have been addressed in the paper and what its proposed approach is to do so.

Despite some similarities between these research papers and ours, there are significant differences between them, which makes them uncomparable with our work. The most similar works to ours are [31,35,42]; in [35,42], a cloud-based SDN context is considered, and two separate problem-specific algorithms are designed to solve controller node placement, one for leader-based synchronization and the other for leaderless synchronization. Hence, since we aim to address synchronization cost in general, regardless of the strategy, they do not apply to our problem. In addition, the proposed approach in [35] is a heuristic and problem-specific algorithm in a wired SDN context, whose main focus is on dynamic controller placement and reassignment of switches to controllers; thus, it is not applicable to our problem either.

4. Problem description

In this paper, we assume an SDN-enabled sensor network, where sensor nodes collect information and forward it towards sink nodes. We assume that the placement of the sink nodes has been already accomplished using the state-of-the-art methods such as [5,12], and here, we merely focus on the placement of controller nodes with respect to the given location of sinks and sensors. Additionally, some other assumptions in this paper are as follows:

- SDN controller nodes are resourceful devices, which are capacitated enough to run the SDN node placement algorithms.
- The controller nodes are supposed to be on-site—that is, no cloud layer is considered.
- The controller plane in SDN is considered to be physically distributed, but logically centralized. To achieve this, controllers need to have a consistent view of the network through constant synchronization.
- Since, in several controller placement application domains, such as IIoT and smart factory, most machinery and robots are fixed and stationary, in this paper, we only consider fixed and stationary sensors. However, in our future work, we will consider mobile devices and sensors.

It is also worth mentioning that throughout this paper, the terms $latency$ and $delay$ are used interchangeably. Moreover, there are a wide range of factors which can potentially contribute to network latency [43], including but not limited to: queuing delay–the amount of time a packet waits in a queue before being further processed; the transport layer delay–such as TCP handshake delay, loss recovery delay, and network congestion control delay, in case of using TCP protocol as transport layer protocol; transmission and propagation delay; traffic load–the amount of data moving across each link at a given point of time. Despite such variety, we consider traffic load as the major delay-contributing factor and are going to take other factors into account in our future work.

4.1. Problem representation

A WSN is represented by an undirected graph, where vertices are partitioned into a set of sensors T that continuously generate data, sinks S to collect the sensors' data, and controllers C to implement the SDN functionalities. Consequently, in the graph representing a WSN, $V = T \cup S \cup C$. An edge of the graph denotes a wireless connection between a pair of nodes. The total number of sensors is N, where $N = |T|$. Two nodes are considered to be *neighbors* if and only if there exists an edge that makes them connected. Furthermore, to represent the placement problem, we define A_C as a set of candidate controllers. Hence, $C \subseteq A_C$.

To represent the problem, a binary vector X is utilized which determines whether a candidate controller has been selected or not. Thus, the size of X equals to the number of candidate controllers:

$$X_i = \begin{cases} 1 & \text{if the candidate controller } i \text{ is chosen} \\ 0 & \text{else} \end{cases} \tag{1}$$

4.2. Reliability constraints

A sensor is k-controller-covered if and only if it has at least k paths of length $\leq l_{max}$ to k controllers in C (k is an integer number ≥ 1). A network is defined as k-controller-covered if each sensor $v \in T$ is k-controller-covered. In order to represent the k-controller-covered constraint, we define the binary matrix Y that determines if the shortest distance between the ith sensor and the jth candidate controller is shorter than l_{max} or not:

$$Y_{i,j} = \begin{cases} 1 & l^*(v_i, c_j) \leq l_{max} \\ 0 & \text{else} \end{cases} \tag{2}$$

where $l^*(v_i, c_j)$ is the shortest path (in terms of the number of hops) between node v_i and controller c_j, which can be calculated by the Dijkstra algorithm.

It is expected that each sensor is covered by k controllers ($k \geq 1$) to avoid a single point of failure and to have a reliable network. Now we can formulate the k-controller-covered constraint as follows:

$$\sum_{\forall c_j \in A_C} Y_{i,j} X_j \geq k; \forall v_i \in T \tag{3}$$

where v_i denotes the ith sensor.

To represent k-controller-covered constraint for all sensors using a single formula, Eq. (3) is rewritten as:

$$P_1(X) = \sum_{\forall v_i \in T} \max\left\{0, \left(k - \sum_{\forall c_j \in A_C} Y_{i,j} X_j\right)\right\} \leq 0 \tag{4}$$

4.3. Timing constraints

To meet timing requirements in WSNs, the rate of routing requests coming from sensors to a particular controller should not exceed a threshold. The threshold denotes the maximum rate of routing requests that a controller can process in an acceptable time [27,44]. In other words, if the load of a controller exceeds the threshold, the controller is overloaded. This threshold is considered in our model and called the load constraint. In the formulation of the load constraint, the load of sensor i is denoted by ω_i and defined as the rate of routing messages per second for those routing messages that do not match the sensor's lookup table and must be sent to the controller [27]. If a sensor is covered by n controllers (i.e., the distance between those n controllers and the sensor is shorter than l_{max}), the load of the ith sensor on the jth controller covering the sensor is denoted by $w_{i,j}$, which is equal to ω_i/n. In other words, we assume that the load of a sensor is uniformly distributed across the controllers covering the sensor [12]. Let us assume that all the controllers have the same type and the same capacity, and W

Table 1
Related work on controller placement (see [31,32,35–42]).

	Context	Objectives			Method
		Sensor-controller delay	Inter-controller delay	Synchronization overhead	
[36]	Wireless SDN	✓	✓	×	A heuristic approach based on Dijkstra and K-means algorithm
[37]	Wireless SDN	✓	✓	×	Garter Snake Optimization algorithm
[38]	Wireless SDN	✓	✓	×	Salp Swarm Optimization algorithm
[39]	Wireless SDN	✓	✓	×	Particle Swarm Optimization algorithm
[40]	Wireless SDN	✓	✓	×	Firefly algorithm and Particle Swarm Optimization algorithm
[41]	Wireless SDN	✓	✓	×	Integer Programming
[32]	Wired SDN	×	✓	✓	Integer Quadratic Programming and a heuristic algorithm
[35] [42]	Cloud-based SDN	✓	×	✓	Linearization and submodularity techniques for small-scale networks; two separate algorithms to address leader-based and leaderless synchronization for large-scale networks.
[31]	Wired SDN	✓	✓	✓	Integer Linear Programming and two heuristic algorithms based on Simulated Annealing and Greedy Knapsack to address dynamic placement of controllers and reassignment of switches to controllers.

shows the maximum workload that a controller can handle within an acceptable time, then the workload constraint is reflected by:

$$\sum_{\forall v_i \in T} Y_{i,j} X_j \frac{\omega_i}{\sum_{\forall c_t \in A_C} Y_{i,t} X_t} \leq W/(k-1); \forall c_j \in A_C \tag{5}$$

where $\sum_{\forall c_t \in A_C} Y_{i,t} X_t$ is the number of selected controllers covering the ith sensor. Due to reliability requirements, we apply a stricter workload constraint; if $k-1$ controllers fail, the remaining active ones should be able to handle the entire workload of all the failed controllers without any violation of the latency requirements. That is the reason to divide the right side of the inequality by $k-1$. In other words, if the workload of a controller exceeds this threshold, its task might be delayed and negatively affects the timeliness of the network. To formulate load constraint for all controllers, Eq. (5) is transformed into:

$$P_2(X) = \sum_{\forall c_j \in A_C} \max\left\{0, \left(\sum_{\forall v_i \in T} Y_{i,j} X_j \frac{\omega_i}{\sum_{\forall c_t \in A_C} Y_{i,t} X_t} - \frac{W}{k-1}\right)\right\} \leq 0 \tag{6}$$

It is worth mentioning that besides the load constraint, the limitation of the number of hops between sensors and controllers reflected in the definition of k-controller-covered is a sort of timing constraint; otherwise, having only k controllers would be enough even for huge WSNs.

4.4. Budget constraints

We assume all the controllers have the same type and the same price, denoted by $Price^{controller}$. We have a certain budget to purchase controller nodes which is denoted by *Budget*. Obviously, the purchasing cost of controllers should not exceed the given budget.

$$\sum_{\forall c_j \in A_C} X_j \leq \left\lfloor \frac{Budget}{Price^{Controller}} \right\rfloor \tag{7}$$

4.5. Inter-controller synchronization cost

As mentioned previously, although the scattered placement of controllers would significantly reduce the delay of communication between sensors and controllers, it would increase inter-controllers synchronization cost.

To evaluate the synchronization cost, two metrics are considered to be of paramount importance: synchronization delay (in terms of the number of hops between controllers) and synchronization overhead [4, 31]. These two metrics are defined as follows:

- Synchronization delay: The time it takes that a controller becomes aware of a state change in another controller. It is measured in terms of the number of hops between two controllers.
- Synchronization overhead: The number of exchanged inter-controller messages for state synchronization.

To represent the synchronization overhead between each pair of controllers, according to the state-of-the-art, including [4,31], we define matrix m in which $m_{i,j}$ is the number of exchanged synchronization messages between controller c_i and c_j. Thus the synchronization cost, that is denoted by *Sync*, can be formulated as follows:

$$Sync = \sum_{\forall c_i \in A_C} \sum_{\forall c_j \in A_C} l^*(c_i, c_j)\ m_{i,j}\ X_i X_j \tag{8}$$

where $l^*(c_i, c_j)$ is the synchronization delay in terms of the number of hops between two controllers, c_i and c_j.

4.6. Optimization problem formulation

Since the controller needs to keep in touch with sensors to dynamically manage the routing decisions, the number of hops between sensors and the controller(s) considerably affects the network performance. A long path between sensors and the controller(s) can increase not only the network traffic but also the delay of exchanging control messages. Accordingly, to improve the sensor-controller communication delay, we would like to place the controller nodes such that the farthest controller that covers a sensor becomes as close as possible to the sensor. Hence, to minimize the maximum distance between sensors and controllers we should:

$$Minimize: \quad \max_{\forall v_i \in T} \{L^*_{v_i}\} \tag{9}$$

which $L^*_{v_i}$ denotes the distance between v_i and its furthest controller among k controllers which covers it:

$$L^*_{v_i} = \max_{\forall c_j \in A_c} \{Y_{i,j} X_j l^*(v_i, c_j)\} \tag{10}$$

Apart from that, distributed control logic arises another issue. To maintain a consistent global state, the controllers need state synchronization which incurs extra overhead. Thus, although using more controllers can potentially reduce sensor-controller communication delay, it would increase inter-controller synchronization cost.

Hence, to study the trade-off between sensor-controller communication delay and inter-controller synchronization cost, the optimization problem is formulated as follows:

$$\begin{aligned} &Minimize: \quad \alpha\ Sync + \beta \sum_{\forall v_i \in T} \{L^*_{v_i}\} \\ &Subject\ to: \quad (4), (6), (7) \end{aligned} \tag{11}$$

In Eq. (11), α and β are constants and are used to adjust the relative importance of sensor-controller communication delay and inter-controller synchronization cost.

4.7. Fitness function

In the previous section, we modeled our problem as a binary Integer Linear Problem (ILP). Since this problem is NP-hard, finding the global optimum solution using exhaustive search is infeasible. Alternatively, we use meta-heuristic algorithms to find near optimum in reasonable execution time.

In addition, as we are going to apply meta-heuristic algorithms, to be able to evaluate how close a solution is to the global optimum or how much a solution violates constraints, the goal function and the constraints are integrated into a single fitness function:

$$Fitness(X) = \alpha\ Sync + \beta \sum_{\forall v_i \in T} \{L^*_{v_i}\} + \gamma \left(P_1(X) + P_2(X)\right) \tag{12}$$

In Eq. (12), P_1 and P_2 are penalty functions used to measure the satisfiability of each solution. A given solution meets all constraints only if both P_1 and P_2 functions equal to zero; otherwise, at least one of the constraints is violated.

5. Solution framework

Since the problem is NP-hard [45,46], the time complexity of computing the exact optimal solution is exponential, and finding the global optimum is practically intractable. Accordingly, to provide an acceptable compromise between the quality of solutions and the runtime, heuristic algorithms are needed. In this paper, we use the Cuckoo optimization algorithm, an evolutionary meta-heuristic algorithm that is inspired by nature.

5.1. Cuckoo optimization algorithm

Many meta-heuristic algorithms are inspired by nature and have been successfully applied in a wide range of optimization problems. One of these algorithms is Cuckoo Search (CS) which is developed by Xin-She Yang and Suash Deb in 2009 [47]. This algorithm seeks to find global optimization by imitating brood parasitism of some cuckoo species. Brood Parasitism of cuckoos means these birds never build their own nests and instead lay their eggs in the nest of other host birds which just laid its eggs. If the host bird discovers that the egg is not its own, it will throw the egg away [48]. Otherwise, the cuckoo eggs hatch a little earlier than the host eggs, and the cuckoo chicks may evict the host eggs out of the nest and this causes an increase in cuckoos population. Thus, for cuckoos, finding a suitable habitat is key to survive. [48,49]

The cuckoo search algorithm is an evolutionary and memory-based optimization algorithm. The pseudo-code of this algorithm is presented in Algorithm 1. This algorithm starts from an initial set of mature cuckoos which lay some eggs. Each mature cuckoo represents a solution, and each egg represents a new solution. Note that each mature cuckoo can lay from 5 to 20 eggs within a maximum distance from their habitat, which is called "Egg Laying Radius" (ELR). *ELR* for each cuckoo is proportional to the total number of eggs, the number of the cuckoo's eggs, the upper bound and the lower bound for the variable. The formula to calculate *ELR* is [50]:

$$ELR = \alpha \times \left(\frac{Number\ of\ Current\ Cuckoo's\ Eggs}{Total\ Number\ of\ Eggs}\right) \times \left(var_{hi} - var_{low}\right) \tag{13}$$

Furthermore, for each egg, a profit value is calculated that represents the probability of having the chance to grow. From all eggs, about p percent of them (usually 10 percent) with less profit value will be detected and killed by the host. Other eggs have the chance to grow and become a mature cuckoo, which causes an increase in the total population of cuckoos. However, since there is always an equilibrium in the bird's population, the total population should not exceed from a maximum number called N_{max}. Therefore, at the end of each iteration, if the total population of mature cuckoos exceeds N_{max}, about q percent of mature cuckoos with less profit, are killed [50].

The cuckoo search algorithm has been successfully applied in several optimization problems such as job scheduling [51], production planning problem [52], and precedence constrained sequencing [53].

It is worth mentioning that, since cuckoo search uses no gradient information during the search process, it has the ability to solve non-convex, nonlinear, nondifferentiable, and multimodal problems [54]. It has also been successfully applied in several nonlinear optimization problems, namely in [55] and [56].

5.2. Applying cuckoo to our placement problem

As we use the cuckoo algorithm to solve the synchronization-aware controller placement in WSNs, the proposed method is called SYNCOP, wherein:

- Each cuckoo or each egg is a representation of X.
- The profit value of each cuckoo or egg is calculated according to Eq. (12).
- ELR is the count of all X items which can be toggled from each mature cuckoo to its eggs.

Some further details regarding the framework configuration are described below:

- *Problem space:* It is the set of all possible selections from the set of candidate controllers. Therefore, each point in problem space can be represented using an array of binary values with size $|A_C|$.
- *Solution representation:* Each mature cuckoo and each egg represents a point in the problem space as a potential solution.
- *Neighborhood structure:* Each mature cuckoo can lay from 5 to 20 eggs. These eggs are a subset of all reachable points by each mature cuckoo.
- *Generating a neighbor:* To generate neighbors of a point in problem space, we use ELR which is calculated according to Eq. (13). In other words, the distance between each point in the problem space with its neighbors should be less than the ELR. Furthermore, it is assumed that the distance between two points in problem space is calculated as the number of all X toggled items.
- *Fitness function:* To evaluate each point in problem space, we use fitness function which is described in Eq. (12).

6. Performance evaluation

6.1. Experimental setups

Our experiments are conducted on four benchmarks, each of which is a WSN with different sizes. The first benchmark includes 100 sensors and 20 candidate locations to place controllers and is referred to as WSN1. The second benchmark consists of 150 sensors and 30 candidate locations to place controllers and is called WSN2. The third benchmark, named as WSN3, has 200 sensors and 40 candidate locations to place controllers. Finally, the last benchmark includes 300 sensors and 50 candidate locations to place controllers and is referred to as WSN4. Other system parameters are derived from state-of-the-art including [26] and [57] which are listed in Table 2.

We have implemented both SA and QA as well as SYNCOP in Java, and the source code is publicly available in [58]. The algorithms were executed on a laptop with Mac OS, Core i5 2.9 GHz, and 8 GB memory. All of them were run 30 times to reach a 95% confidence interval. In addition, we have used CPLEX ILP Solver and set its settings and parameters to default.

Algorithm 1: Cuckoo optimization algorithm

Calculate $|C|$, the number of selected controllers, according to Eq. (7);
Initialize α and β in Eq. (11);
Initialize the population: N_{pop} host nests (solution) $x_i (i = 1..N_{pop})$;
Calculate the fitness value for the initial solutions: $F_i, i = 1, 2, ..., N_{pop}$;
repeat
  for $i = 1$ **to** N_{pop} **do**
    Generate cuckoo egg x_k (new solution) by using Levy flights;
    Step 1: Calculate reliability constraint fitness value for x_k as $P_1(x_k)$ (Eq. (4));
    Step 2: Calculate timing constraint fitness value for x_k as $P_2(x_k)$ (Eq. (6));
    Step 3: Calculate $\max_{\forall v_i \in T}\{L^*_{v_i}\}$ for x_k as $maxL$;
    Step 4: Calculate $Sync$ for x_k (Eq. (8));
    Evaluate fitness value for x_k based on P_1, P_2, $maxL$, and $Sync$ (Eq. (12));
    Choose nest j among N_{pop} randomly;
    if $F_k > F_j$ **then**
      Replace solution j by the new solution;
    end
    Step 5: Sort nests by their fitness values;
    A fraction p of worst nests are abandoned and new ones are built;
    Keep the best solutions or nests;
  end
  if *Cuckoo Population* $> N_{max}$ **then**
    Step 6: Sort cuckoos by their fitness values;
    Kill q percent of cuckoos with less fitness value;
  end
until $t < MaxGenerations$ *or Stop Criteria*;

Table 2
System parameters.

Benchmark	Sensors	Candidate controllers	Selected controllers	Budget	Controllers cost	k-covered	Workload capacity
WSN1	100	20	6	3000$	500$	6	30 Bytes/s
WSN2	150	30	10	5000$	500$	6	30 Bytes/s
WSN3	200	40	13	6500$	500$	6	30 Bytes/s
WSN4	300	50	16	8000$	500$	6	30 Bytes/s

According to the formulated optimization problem presented in Eq. (11), we define several metrics to evaluate the performance of the algorithms. The first metric is the sensor-controller communication delay, in terms of the summation of L^* for all nodes, denoted by ϕ, and the second metric is the inter-controller synchronization cost, denoted by $Sync$ and is calculated based on Eq. (8). The defined metrics are presented in Table 3. We have also taken execution time into account and have compared SYNCOP against other algorithms in this regard. Moreover, as discussed previously in Sections 1 and 2, some critical requirements in the controller placement problem are inter-controller synchronization cost, sensor-controller communication delay, and execution time. Hence, if SYNCOP is able to outperform the other algorithms in terms of these metrics and satisfy these requirements, it can be considered an efficient algorithm to address the controller placement problem.

6.2. Sensitivity analysis

To evaluate the effectiveness of the algorithm parameters on the performance of the algorithm, a sensitivity analysis is performed. The sensitivity analysis is often employed to assess the extent to which independent variables would have an impact on dependent variables. To do so, a reasonable range for every parameter of the algorithm is investigated to find the best values. As a result, algorithm parameters are listed in Table 4 in which the name of each parameter, its evaluated range, and its best value are shown in the first, second, and third columns, respectively. It is worth mentioning that the proper values for QA and SA parameters are selected based on [59–62]. Additionally, the parameter ranges for the SYNCOP algorithm are chosen based on [63–65]. These references are presented in the fourth column of Table 4.

Since the parameters of SA and QA, which are mentioned in Table 4, may seem to be a little vague, let us demystify them using a brief explanation of each of these algorithms and each one's parameters. First, about SA algorithm and its parameters; *Annealing* is a physical process in which a material is heated and then gradually cooled until it reaches a specific temperature. SA (Simulated Annealing) is a meta-heuristic algorithm, which models this process and is used to approximate the global optimum in a large search space. In this regard, T_0 refers to the initial temperature, T_f specifies until when the process should be continued, and μ determines the rate of decreasing the temperature, which is also called *cooling* process. Second, about QA algorithm and its parameters; QA (Quantum Annealing) is also a meta-heuristic algorithm, which comes from quantum mechanics and seeks to approximate the global optimum in a large search space with several high barriers. The technique with the help of which QA has the ability to escape from local optima is called *Quantum Tunneling*. Quantum tunneling measures to what extent the algorithm is able to jump over a barrier. At first, this parameter is set to a high number so that the searching ability of the algorithm is significantly high at the beginning; by the passage of time, the quantum tunneling parameter is reduced at a smooth rate. In this respect, Γ_0 refers to the initial tunneling value, Γ_f specifies the final tunneling value, σ is the rate of decreasing the tunneling value, T plays the same role as in SA, and

Table 3
Evaluation metrics.

Metric	Description	
$Sync$	Inter-controller synchronization cost	Eq. (8)
ϕ	Sensor-controller communication delay	$\sum_{\forall v_i \in T}\{L^*_{v_i}\}$

finally, tr is the number of replicas in the system—the more the number of replicas is, the more accurate solution is achieved at the expense of consuming more memory and taking longer time.

6.3. Time complexity

Time complexity is a promising approach to measure how a change in the input size can affect the execution time of an algorithm. Hence, in this section, a comprehensive analysis of the time complexity of the SYNCOP algorithm is presented.

SYNCOP, as presented in Algorithm 1, is composed of six main steps. A brief explanation of each step and its relevant time complexity is as follows:

- Step 1: The first step is to calculate the reliability constraint fitness value, according to Eq. (4). The time complexity of this step can be analyzed as: $O\left(|T||A_C|\right)$.
- Step 2: The second step is to calculate the timing constraint fitness value, according to Eq. (6). This step has time complexity: $O\left(|T||A_C|\right)$.
- Step 3: The third step is to calculate the maximum distance between sensors and controllers, according to Eq. (10). The time complexity of this step is: $O\left(|T||A_C|\right)$.
- Step 4: The fourth step is to calculate synchronization cost using Eq. (8). The time complexity of this step is: $O\left(|A_C|^2\right)$
- Step 5: The fifth step is to sort all eggs by their fitness values. Since according to Table 4, each mature cuckoo has at most 20 eggs, the time complexity of this step can be written as $O\left(20 \times N_{pop} \log N_{pop}\right)$
- Step 6: The sixth step is to sort mature cuckoos by their fitness values, whose time complexity is: $O\left(N_{pop} \log N_{pop}\right)$

Hence, the time complexity can be written as $O\left(N_{pop} \log N_{pop} + |T||A_C| + |A_C|^2\right)$. Moreover, since increasing the network size does not result in a considerable change in N_{pop}, it can be concluded that $|T||A_C| + |A_C|^2$ has dominance over $N_{pop} \log N_{pop}$. Thus, the overall time complexity of SYNCOP is: $O\left(|T||A_C| + |A_C|^2\right)$.

6.4. Results

In this section, we will illustrate and analyze our experimental results. These results provide insights about the trade-off between inter-controller synchronization cost and sensor-controller communication delay.

Impact of Weights α and β. As mentioned in Eq. (11), since there is a trade-off between sensor-controller communication delay and inter-controller synchronization cost, two constants are used as weights to adjust the relative importance of these two objectives. These two constants are α and β, which are respectively, the weight of inter-controller synchronization cost and sensor-controller communication delay, where $\alpha + \beta = 1$. Therefore, by setting α to 1 (obviously $\beta = 0$), the optimization objective becomes to reduce the inter-controller synchronization cost, neglecting the sensor-controller communication delay at all. However, by decreasing the α, hence increasing β value, more priority is given to the sensor-controller communication delay. Similarly, when $\alpha = 1$ (obviously $\beta = 0$), the main focus is to reduce the inter-controller synchronization cost, and thus the sensor-controller communication delay is totally neglected. On the other hand, it is vital to highlight the fact that, when α is less than 0.5 (i.e., β is more than 0.5), sensor-controller communication delay is preferred over inter-controller synchronization cost. However, when α is between 0.5 and 1 (i.e., β is between 0 and 0.5), inter-controller synchronization cost is preferred over sensor-controller communication delay.

Interaction Between Different Objectives. Fig. 1 shows the results for WSN1, wherein Fig. 1(a) represents the inter-controller synchronization cost, and Fig. 1(b) shows the sensor-controller communication delay. In both graphs, the x-axis depicts the different values in α range. As mentioned previously, since it is assumed that $\alpha + \beta = 1$, representing the α value suffices. In addition, the colored lines represent different algorithms. Similarly, the results for WSN2, WSN3, and WSN4 are depicted in Fig. 2, Fig. 3, and Fig. 4, respectively. It is worth mentioning that, due to extremely long execution time of ILP, it is infeasible to apply ILP in WSN4 (very large network size). Hence, in Fig. 4, which depicts the results for WSN4, only the results of SA, QA, and SYNCOP are displayed.

Apart from that, Table 5 shows the improvement percentage of SYNCOP against SA and QA in all benchmarks, for different values of weight α. In addition, the average improvement percentage of SYNCOP against SA and QA for all benchmarks are represented in Table 6. It is also worth mentioning that the formula with the help of which the improvement percentages are calculated is: $\frac{New\ value - Old\ value}{Old\ value} \times 100$. In addition, the value of fitness function for SYNCOP, which is calculated using Eq. (12) is represented in Table 7.

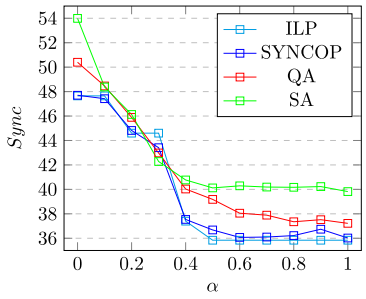
Further details on the results for all benchmarks are provided in the following sections.

As shown in Figs. 1–4, by increasing the α value (i.e., giving more priority to inter-controller synchronization cost), for all employed methods (i.e., ILP, SYNCOP, SA, and QA), there is a gradual upward trend in sensor-controller communication delay. At the same time, there is a dramatic decrease in inter-controller synchronization cost. Therefore, our findings for different network sizes, support the notion that inter-controller synchronization cost and sensor-controller communication delay are two contradicting objectives. In other words, although the scattered placement of controllers in an SDN-enabled WSN would probably decrease the sensor-controller communication delay, it will have a negative effect on the inter-controller synchronization cost.

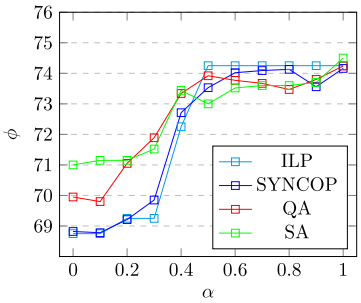
Furthermore, when the sensor-controller communication delay is preferred (i.e., α is between 0 and 0.5), SYNCOP outperforms QA and SA in ϕ for all α values. However, in the same range of α, in terms of $Sync$, SYNCOP falls behind in some cases. In other words, when sensor-controller communication delay is preferred, SYNCOP, in comparison to SA and QA, is able to further decrease the sensor-controller communication delay by tolerating some additional synchronization cost, which seems to be a normal behavior. Therefore, when α is between 0 and 0.5, although the solution achieved by SYNCOP may have worse $Sync$, it will always be better in terms of fitness value. This behavior is a piece of evidence for negative improvement percentages for $Sync$ when sensor-controller communication is preferred.

On the other hand, when inter-controller synchronization cost is preferred (i.e., α is between 0.5 and 1), the SYNCOP results are substantially better than QA and SA in $Sync$. However, SYNCOP underperforms QA and SA in terms of ϕ in some cases. In other words, when inter-controller synchronization cost is preferred, SYNCOP, in comparison to QA and SA, always finds better fitness value, even at the expense of worsening sensor-controller communication delay, which is an expected behavior.

Pareto Optimality Analysis. As our findings support the notion that inter-controller synchronization cost and sensor-controller communication delay are two contradicting objectives, there is no single solution that simultaneously minimizes each of these two objectives. In other words, there is a set of solutions in which none of the objectives can be improved without degrading the other. Each of these non-dominated

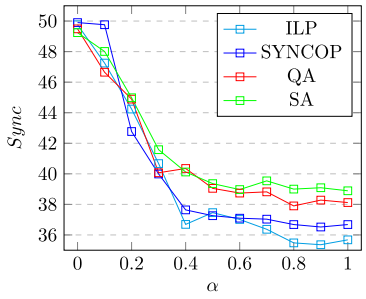


(a) Amount of $Sync$ for different α weights

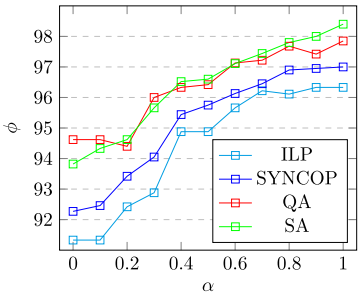


(b) Amount of ϕ for different α weights

Fig. 1. Results for WSN1.

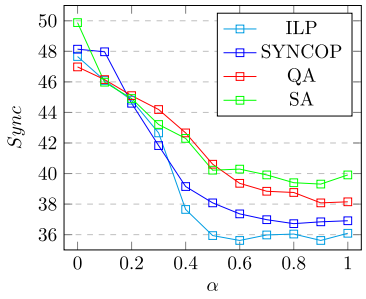


(a) Amount of $Sync$ for different α weights

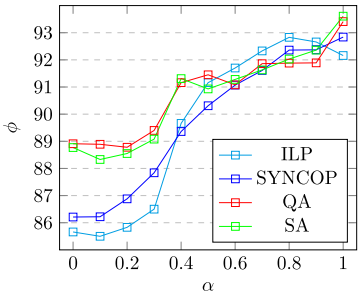


(b) Amount of ϕ for different α weights

Fig. 2. Results for WSN2.

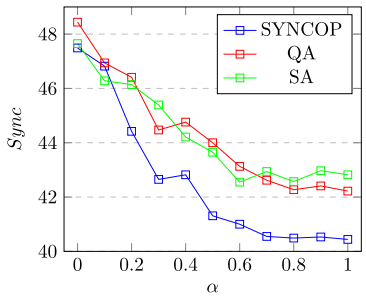


(a) Amount of $Sync$ for different α weights

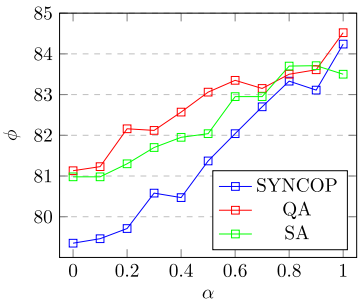


(b) Amount of ϕ for different α weights

Fig. 3. Results for WSN3.



(a) Amount of $Sync$ for different α weights



(b) Amount of ϕ for different α weights

Fig. 4. Results for WSN4.

Table 4
Algorithm parameters.

Algorithm	Parameter	Description	Evaluated range	Best value	References
SA	T_0	Initial temperature	50–150	100	[59,60]
	T_f	Final temperature	0.1–1	0.5	
	μ	Temperature cooling rate	0.75–0.99	0.9	
	N_{SA}	Monte Carlo steps	50–120	100	
QA	tr	Trotter replica	50–120	100	[59,61,62]
	Γ_0	Initial tunneling field	0.5–2	1	
	Γ_f	Final tunneling field	0.1–0.5	0.5	
	σ	Tunneling field reduction rate	0.7–0.99	0.95	
	T	Temperature	30–100	50	
	N_{QA}	Monte Carlo steps	50–120	110	
SYNCOP	N_{pop}	Initial population	50–200	100	[63–65]
	N_{max}	Max population	500–1500	1000	
	p	Eggs killing rate (By hosts)	0.1–0.5	0.2	
	q	Mature cuckoo killing rate	0.1–0.5	0.2	
	$eggs$	Number of eggs per cuckoo	–	5–20	

Table 5
Improvement percentage of SYNCOP against other baselines.

	α	β		WSN1		WSN2		WSN3		WSN4	
				SA	QA	SA	QA	SA	QA	SA	QA
More priority is given to ϕ	0	1	$Sync$	11.65%	5.36%	−1.34%	−0.83%	3.49%	−2.47%	0.34%	1.96%
			ϕ	3.07%	1.62%	1.65%	2.48%	1.78%	3.04%	2.01%	2.19%
	0.1	0.9	$Sync$	1.98%	2.15%	−3.65%	−6.67%	−4.33%	3.94%	−1.17%	0.26%
			ϕ	3.33%	1.46%	1.98%	2.28%	2.39%	3.0%	1.88%	2.18%
	0.2	0.8	$Sync$	2.84%	2.33%	4.9%	4.72%	0.69%	1.11%	3.73%	4.29%
			ϕ	2.73%	2.59%	1.27%	1.04%	1.89%	2.14%	1.96%	2.98%
	0.3	0.7	$Sync$	2.67%	1.0%	3.82%	0.2%	3.19%	5.34%	6.04%	4.09%
			ϕ	2.34%	2.84%	1.68%	2.03%	1.39%	1.74%	1.37%	1.88%
	0.4	0.6	$Sync$	7.92%	6.22%	6.18%	6.72%	7.45%	8.23%	3.14%	4.33%
			ϕ	0.99%	1.13%	1.12%	0.92%	2.14%	1.97%	1.81%	2.54%
Both $Sync$ and ϕ have equal priority	0.5	0.5	$Sync$	8.57%	6.38%	5.34%	4.58%	5.32%	6.23%	5.36%	6.11%
			ϕ	−0.7%	0.53%	0.88%	0.69%	0.68%	1.25%	0.82%	2.03%
More priority is given to $Sync$	0.6	0.4	$Sync$	10.47%	5.2%	4.82%	4.26%	7.27%	5.08%	3.64%	4.94%
			ϕ	−0.68%	−0.34%	1.01%	1.03%	0.23%	0.01%	1.1%	1.57%
	0.7	0.3	$Sync$	10.18%	4.72%	6.35%	4.61%	7.34%	4.79%	5.57%	4.86%
			ϕ	−0.67%	−0.57%	1.02%	0.79%	0.02%	0.27%	0.3%	0.54%
	0.8	0.2	$Sync$	9.83%	3.03%	5.95%	3.22%	5.94%	5.26%	4.89%	4.21%
			ϕ	−0.72	−0.9%	0.92%	0.8%	−0.36%	−0.58%	0.44%	0.2%
	0.9	0.1	$Sync$	8.68%	2.08%	6.57%	4.6%	6.28%	3.26%	5.68%	4.43%
			ϕ	0.19%	0.33%	1.07%	0.48%	−0.01%	−0.52%	0.72%	0.59%
	1	0	$Sync$	9.54%	5.76%	5.68%	3.78%	7.52%	3.25%	5.56%	4.22%
			ϕ	0.46%	0.11%	1.42%	0.87%	0.61%	0.82%	−0.89%	0.33%

Table 6
Average improvement percentage of SYNCOP against other baselines.

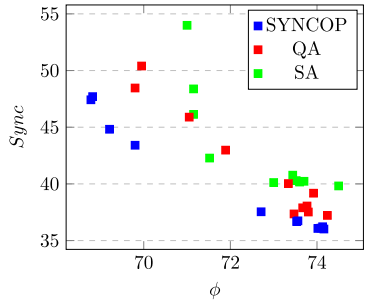
		WSN1		WSN2		WSN3		WSN4	
		SA	QA	SA	QA	SA	QA	SA	QA
More priority is given to ϕ	$Sync$	5.41%	3.41%	1.98%	0.828%	2.09%	3.23%	2.41%	2.98%
	ϕ	2.49%	1.93%	1.54%	1.75%	1.92%	2.37%	1.81%	2.35%
	Overall	3.95%	2.67%	1.76%	1.29%	2.01%	2.8%	2.11%	2.66%
Both $Sync$ and ϕ have equal priority	$Sync$	8.57%	6.38%	5.34%	4.58%	5.32%	6.23%	5.36%	6.11%
	ϕ	−0.7%	0.53%	0.88%	0.69%	0.68%	1.25%	0.82%	2.03%
	Overall	3.94%	3.81%	3.11%	2.64%	3%	3.74%	3.09%	4.07%
More priority is given to $Sync$	$Sync$	9.74%	4.16%	5.87%	4.09%	6.87%	4.33%	5.07%	4.53%
	ϕ	−0.28%	−0.27%	1.09%	0.79%	0.1%	0%	0.33%	0.65%
	Overall	4.73%	1.95%	3.48%	2.44%	3.44%	2.17%	2.7%	2.59%
Overall average	$Sync$	7.91%	4.65%	4.40%	3.16%	4.76%	4.6%	4.28%	4.54%
	ϕ	0.50%	0.73%	1.17%	1.08%	0.9%	1.21%	0.99%	1.68%
	Overall	4.21%	2.81%	2.78%	2.12%	2.82%	2.9%	2.63%	3.11%

solutions is called Pareto-optimal, and the set of all Pareto-optimal solutions is called Pareto-front.

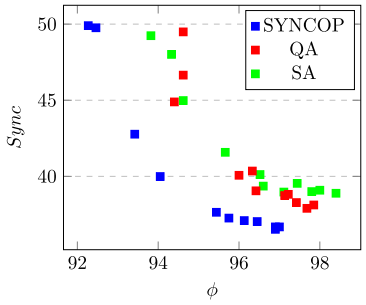
To look at the trade-off between inter-controller synchronization cost and sensor-controller communication delay in more detail and to compare the results of SYNCOP against the other two algorithms, we leveraged Fig. 5. In Fig. 5, each point represents a placement with a given alpha and beta. The x-value and y-value of each point indicate the corresponding sensor-controller communication delay and inter-controller synchronization cost, respectively.

Table 7
Fitness function values for SYNCOP.

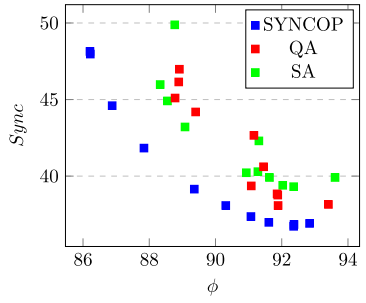
Benchmarks	Fitness	α											Average
		0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	
WSN1	Worst	69.25	66.78	64.53	62.07	55.44	58.88	51.54	48.81	45.08	41.54	37.85	54.70
	Average	68.82	66.65	64.33	61.91	55.11	58.64	51.25	47.50	43.80	40.43	36.02	54.04
	Best	68.75	66.54	64.15	61.85	55.04	58.61	51.20	47.36	43.52	39.68	35.84	53.86
WSN2	Worst	92.88	88.45	83.57	78.45	72.70	66.97	61.09	55.26	49.29	43.35	37.16	66.29
	Average	92.27	87.94	83.29	77.83	72.32	66.51	60.71	54.85	48.73	42.56	36.68	65.79
	Best	91.33	87.17	82.84	77.22	71.61	66.17	60.07	54.32	47.41	41.36	35.68	65.02
WSN3	Worst	86.66	82.64	78.72	74.45	69.61	64.58	59.30	53.77	48.53	43.19	37.56	63.54
	Average	86.21	82.39	78.42	74.04	69.28	64.19	58.85	53.37	47.84	42.40	36.91	63.08
	Best	85.66	81.85	77.62	73.34	68.86	63.55	58.37	52.58	46.9	41.23	36.06	62.36
WSN4	Worst	79.66	76.42	73.00	69.43	65.76	61.66	58.05	53.72	49.33	45.06	41.11	61.2
	Average	79.35	76.19	72.65	69.2	65.41	61.39	57.41	53.19	49.06	44.79	40.44	60.82
	Best	78.88	76.02	72.25	68.92	64.94	60.97	56.88	52.69	48.03	44.29	39.84	60.33



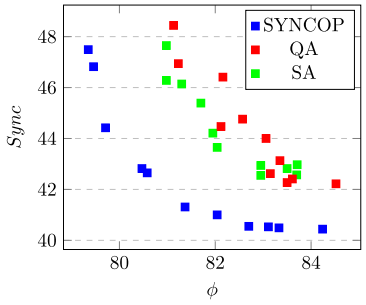
(a) Trade-off between $Sync$ and ϕ for WSN1



(b) Trade-off between $Sync$ and ϕ for WSN2



(c) Trade-off between $Sync$ and ϕ for WSN3



(d) Trade-off between $Sync$ and ϕ for WSN4

Fig. 5. Trade-off between $Sync$ and ϕ for all benchmarks.

As is seen in Fig. 5, since the Pareto-front set mainly consists of solutions found by SYNCOP, it is concluded that SYNCOP outperforms the other two algorithms in both objectives for all benchmarks.

Execution Time. Fig. 6 represents the average execution time, in seconds, for all employed methods in different benchmarks.

In WSN1, although the execution time of SA, QA, and SYNCOP are almost in the same range, ILP takes twice as much as SYNCOP. For other benchmarks, the average execution time of SA and QA are almost in the same range. Although, due to Table 6, SYNCOP has significantly better performance than QA, it is about 0.5%, 0.6%, and 0.7% slower for WSN2, WSN3, and WSN4, respectively.

On the other hand, ILP is noticeably slower than SYNCOP for all benchmarks. Moreover, it is worth mentioning that, due to extremely long execution time of ILP, it is infeasible to apply ILP in WSN4 (very large network size).

Therefore, it leads to the conclusion that SYNCOP can find near-optimal solution in a more reasonable time, in comparison to ILP. Furthermore, since SYNCOP, unlike ILP, is feasible for very large network size, it is scalable.

7. Conclusion

Leveraging SDN technology in WSNs can ease network management. A common approach to improve the performance in an SDN-enabled WSN is to employ multiple controllers to physically distribute the control plane. However, although using multiple controllers would improve network performance, it causes extra synchronization costs. Accordingly, choosing the best controller placement to optimize performance and synchronization cost simultaneously is a challenging research problem.

In this paper, we have formulated the controller placement problem as an optimization problem, considering reliability, fault tolerance, performance in terms of latency, synchronization overhead, and deployment cost. Moreover, we have applied a nature-inspired population-based meta-heuristic algorithm, SYNCOP, to solve the optimization problem. To evaluate our proposed method, we have implemented SYNCOP along with several other state-of-the-art algorithms, namely QA and SA. Also, we considered different benchmarks; from a small scale benchmark, named WSN1 consisting of 100 sensors, to a very

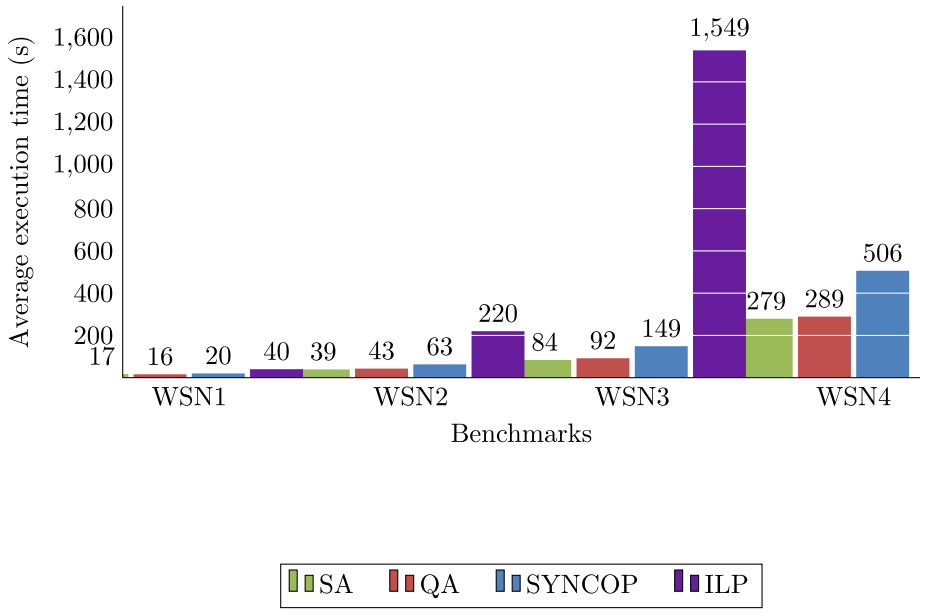


Fig. 6. The average execution time.

large scale one, called WSN4 consisting of 300 sensors. Since in our problem, latency and synchronization overhead are two contradicting objectives, we defined two constant weights to adjust the relative importance of them. Our results reveal that when we set the weights such that more priority is given to synchronization cost, the average improvement of SYNCOP against QA is 4.16%, 4.10%, 4.33%, and 4.53%, for WSN1, WSN2, WSN3, and WSN4, respectively. Furthermore, when more priority is given to performance, the average improvement of SYNCOP against QA is 1.93%, 1.95%, 2.38%, and 2.35%, for WSN1, WSN2, WSN3, and WSN4, respectively. Besides, in terms of scalability, since ILP, due to extremely long processing time, is not scalable for large network size, SYNCOP is capable of finding the near-optimal solution in a far more reasonable time.

In this research, we have leveraged the weighted sum method to convert our multi-objective problem into a single-objective one. However, employing this method comes at the expense of missing a complete Pareto set. Accordingly, as the continuation of this research work, in our future study we intend to use a multi-object algorithm, such as the Multi-Objective Genetic Algorithm (MOGA) or the Non-Dominated Sorting Genetic Algorithm (NSGA). Additionally, in this research, we have conducted our experiments on several general-purpose networks, regardless of considering the routing of them. However, in our future study, we intend to: take into account performing our tests on several real network topologies; consider several other aspects of network latency; consider mobile sensors, such as mobile robots and mobile machinery.

CRediT authorship contribution statement

Shirin Tahmasebi: Methodology, Software, Validation, Investigation, Data curation, Writing - original draft, Visualization, Project administration. **Nayereh Rasouli:** Validation, Investigation, Resources, Visualization. **Amir Hosein Kashefi:** Resources, Writing - original draft. **Elmira Rezabeyk:** Writing - review & editing. **Hamid Reza Faragardi:** Conceptualization, Methodology, Validation, Writing - review & editing, Visualization, Supervision.

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.

References

[1] G. Pereira, R. Alves, D. Alves Galdino de Oliveira, C. Margi, B. Albertini, WS3n: Wireless secure SDN-based communication for sensor networks, Secur. Commun. Netw. 2018 (2018) http://dx.doi.org/10.1155/2018/8734389.

[2] T. Luo, H. Tan, T.Q. Quek, Sensor OpenFlow: Enabling software-defined wireless sensor networks, IEEE Commun. Lett. 16 (2012) 1896–1899, http://dx.doi.org/10.1109/LCOMM.2012.092812.121712.

[3] H. Fotouhi, M. Vahabi, A. Ray, M. Björkman, SDN-TAP: an SDN-Based traffic aware protocol for wireless sensor networks, in: 2016 IEEE 18th International Conference on E-Health Networking, Applications and Services, Healthcom, IEEE, 2016, pp. 1–6.

[4] F. Benamrane, F. Ros, M. Ben Mamoun, Synchronization cost of multi-controller deployments in software-defined networks, Int. J. High Perform. Comput. Netw. 9 (2016) http://dx.doi.org/10.1504/IJHPCN.2016.077821.

[5] H.R. Faragardi, M. Vahabi, H. Fotouhi, T. Nolte, T. Fahringer, An efficient placement of sinks and SDN controller nodes for optimizing the design cost of industrial IoT systems, Softw. Practice Exp. (2018) http://dx.doi.org/10.1002/spe.2593.

[6] A. Dâmaso, N. Rosa, P. Maciel, Reliability of wireless sensor networks, Sensors 14 (9) (2014) 15760–15785.

[7] F. Botelho, A. Bessani, F.M. Ramos, P. Ferreira, On the design of practical fault-tolerant SDN controllers, in: 2014 Third European Workshop on Software Defined Networks, IEEE, 2014, pp. 73–78.

[8] S. Tahmasebi, M. Safi, S. Zolfi, M.R. Maghsoudi, H.R. Faragardi, H. Fotouhi, Cuckoo-PC: An Evolutionary Synchronization-Aware Placement of SDN Controllers for Optimizing the Network Performance in WSNs, Sensors 20 (11) (2020) 3231.

[9] N. McKeown, Software-defined networking, INFOCOM Keynote Talk 17 (2) (2009) 30–32.

[10] B.A.A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, IEEE Commun. Surv. Tutor. 16 (3) (2014) 1617–1634.

[11] H. Mostafaei, M. Menth, Software-defined wireless sensor networks: A survey, J. Netw. Comput. Appl. 119 (2018) 42–56.

[12] H.R. Faragardi, H. Fotouhi, T. Nolte, R. Rahmani, A cost efficient design of a multi-sink multi-controller WSN in a smart factory, in: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems, HPCC/SmartCity/DSS, 2017, pp. 594–602.

[13] F. Derakhshan, S. Yousefi, A review on the applications of multiagent systems in wireless sensor networks, Int. J. Distrib. Sens. Netw. 15 (5) (2019) 1550147719850767.

[14] K. Islam, W. Shen, X. Wang, Wireless sensor network reliability and security in factory automation: A survey, IEEE Trans. Syst. Man Cybern. C 42 (6) (2012) 1243–1256.

[15] M.S. Taboun, R.W. Brennan, An embedded agent-based intelligent industrial wireless sensor network, in: Industrial Applications of Holonic and Multi-Agent Systems, Springer International Publishing, Cham, 2017, pp. 227–239.

[16] M. Bagaa, A. Chelli, D. Djenouri, T. Taleb, I. Balasingham, K. Kansanen, Optimal placement of relay nodes over limited positions in wireless sensor networks, IEEE Trans. Wireless Commun. 16 (4) (2017) 2205–2219.

[17] A.N. Njoya, C. Thron, J. Barry, W. Abdou, E. Tonye, N.S.L. Konje, A. Dipanda, Efficient scalable sensor node placement algorithm for fixed target coverage applications of wireless sensor networks, IET Wirel. Sensor Syst. 7 (2) (2017) 44–54.

[18] S.K. Gupta, P. Kuila, P.K. Jana, Genetic algorithm approach for k-coverage and m-connected node placement in target based wireless sensor networks, Comput. Electr. Eng. 56 (2016) 544–556.

[19] L. Sitanayah, K.N. Brown, C.J. Sreenan, Planning the deployment of multiple sinks and relays in wireless sensor networks, J. Heuristics 21 (2015) 197–232.

[20] W.Y. Poe, J.B. Schmitt, Placing multiple sinks in time-sensitive wireless sensor networks using a genetic algorithm, in: MMB, 2008.

[21] H. Safa, W. El-Hajj, H. Zoubian, A robust topology control solution for the sink placement problem in WSNs, J. Netw. Comput. Appl. 39 (2014) 70–82.

[22] P.C. Srinivasa Rao, H. Banka, P. Jana, PSO-Based multiple-sink placement algorithm for protracting the lifetime of wireless sensor networks, in: Proceedings of the Second International Conference on Computer and Communication Technologies, vol. 379, Springer, New Delhi, 2016, pp. 605–616, http://dx.doi.org/10.1007/978-81-322-2517-1_58.

[23] D. Kim, W. Wang, N. Sohaee, C. Ma, W. Wu, W. Lee, D.-Z. Du, Minimum data-latency-bound *k*-sink placement problem in wireless sensor networks, IEEE/ACM Trans. Netw. 19 (2011) 1344–1353, http://dx.doi.org/10.1109/TNET.2011.2109394.

[24] H.G. Sharma, R. Sharma, Analysis of static and random sink node with different quality of service parameters, in: 2018 International Conference on Advanced Computation and Telecommunication, ICACAT, IEEE, 2018, pp. 1–5.

[25] F. Li, X. Xu, A discrete cuckoo search algorithm for the controller placement problem in software defined networks, in: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference, IEMCON, 2018, pp. 292–296, http://dx.doi.org/10.1109/IEMCON.2018.8614785.

[26] S. Mousavi, S. Fazliahmadi, N. Rasouli, H.R. Faragardi, H. Fotouhi, T. Fahringer, A budget-constrained placement of controller nodes for maximizing the network performance in SDN-enabled WSNs, in: 5th International Conference on Communication, Management and Information Technology, 2019.

[27] A. Dvir, Y. Haddad, A. Zilberman, The controller placement problem for wireless SDN, Wirel. Netw. (2019) http://dx.doi.org/10.1007/s11276-019-02077-5.

[28] Z. Fan, J. Yao, X. Yang, Z. Wang, X. Wan, A multi-controller placement strategy based on delay and reliability optimization in SDN, in: 2019 28th Wireless and Optical Communications Conference, WOCC, IEEE, 2019, pp. 1–5.

[29] M.J. Alenazi, E.K. Çetinkaya, Resilient placement of SDN controllers exploiting disjoint paths, Trans. Emerg. Telecommun. Technol. (2019) e3725.

[30] W. Ren, Y. Sun, H. Luo, M. Guizani, A novel control plane optimization strategy for important nodes in SDN-IoT networks, IEEE Internet Things J. 6 (2) (2018) 3558–3571.

[31] M.F. Bari, A.R. Roy, S.R. Chowdhury, Q. Zhang, M.F. Zhani, R. Ahmed, R. Boutaba, Dynamic controller provisioning in software defined networks, in: Proceedings of the 9th International Conference on Network and Service Management, CNSM 2013, IEEE, 2013, pp. 18–25.

[32] Z. Su, M. Hamdi, MDCP: Measurement-aware distributed controller placement for software defined networks, in: 2015 IEEE 21st International Conference on Parallel and Distributed Systems, ICPADS, IEEE, 2015, pp. 380–387.

[33] K. Choumas, D. Giatsios, P. Flegkas, T. Korakis, The SDN control plane challenge for minimum control traffic: distributed or centralized? in: 2019 16th IEEE Annual Consumer Communications & Networking Conference, CCNC, IEEE, 2019, pp. 1–7.

[34] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, P. Tran-Gia, Pareto-Optimal resilient controller placement in SDN-based core networks, in: Proceedings of the 2013 25th International Teletraffic Congress, ITC, IEEE, 2013, pp. 1–9.

[35] Q. Qin, K. Poularakis, G. Iosifidis, S. Kompella, L. Tassiulas, SDN Controller placement with delay-overhead balancing in wireless edge networks, IEEE Trans. Netw. Serv. Manag. 15 (4) (2018) 1446–1459, http://dx.doi.org/10.1109/TNSM.2018.2876064.

[36] R. Chai, Q. Yuan, L. Zhu, Q. Chen, Control plane delay minimization-based capacitated controller placement algorithm for SDN, EURASIP J. Wireless Commun. Networking 2019 (1) (2019) 1–17.

[37] S. Torkamani-Azar, M. Jahanshahi, A new GSO based method for SDN controller placement, Comput. Commun. 163 (2020) 91–108.

[38] A.A. Ateya, A. Muthanna, A. Vybornova, A.D. Algarni, A. Abuarqoub, Y. Koucheryavy, A. Koucheryavy, Chaotic salp swarm algorithm for SDN multi-controller networks, Eng. Sci. Technol. Int. J. 22 (4) (2019) 1001–1012.

[39] C. Gao, H. Wang, F. Zhu, L. Zhai, S. Yi, A particle swarm optimization algorithm for controller placement problem in software defined network, in: International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2015, pp. 44–54.

[40] K.S. Sahoo, D. Puthal, M.S. Obaidat, A. Sarkar, S.K. Mishra, B. Sahoo, On the placement of controllers in software-defined-WAN using meta-heuristic approach, J. Syst. Softw. 145 (2018) 180–194.

[41] T. Das, M. Gurusamy, Controller placement for resilient network state synchronization in multi-controller SDN, IEEE Commun. Lett. (2020).

[42] Q. Qin, K. Poularakis, G. Iosifidis, L. Tassiulas, SDN Controller placement at the edge: Optimizing delay and overheads, in: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, 2018, pp. 684–692.

[43] X. Zuo, Y. Cui, M. Wang, T. Xiao, X. Wang, Low-latency networking: Architecture, techniques, and opportunities, IEEE Internet Comput. 22 (5) (2018) 56–63.

[44] B. Han, X. Yang, X. Wang, Dynamic controller-switch mapping assignment with genetic algorithm for multi-controller SDN, in: 2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress, DASC/PiCom/CBDCom/CyberSciTech, 2019, pp. 980–986, http://dx.doi.org/10.1109/DASC/PiCom/CBDCom/CyberSciTech.2019.00179.

[45] M. Younis, K. Akkaya, Strategies and techniques for node placement in wireless sensor networks: A survey, Ad Hoc Netw. 6 (4) (2008) 621–655.

[46] A. Efrat, S. Har-Peled, J.S. Mitchell, Approximation algorithms for two optimal location problems in sensor networks, in: 2nd International Conference on Broadband Networks, 2005, IEEE, 2005, pp. 714–723.

[47] X. Yang, Suash Deb, Cuckoo Search via Lévy flights, in: 2009 World Congress on Nature Biologically Inspired Computing, NaBIC, 2009, pp. 210–214, http://dx.doi.org/10.1109/NABIC.2009.5393690.

[48] K.-L. Du, M. Swamy, Search and optimization by metaheuristics, in: Techniques and Algorithms Inspired by Nature, Birkhauser, Basel, Switzerland, 2016.

[49] X.-S. Yang, Chapter 9 - cuckoo search, in: X.-S. Yang (Ed.), Nature-Inspired Optimization Algorithms, Elsevier, Oxford, 2014, pp. 129–139, http://dx.doi.org/10.1016/B978-0-12-416743-8.00009-9, http://www.sciencedirect.com/science/article/pii/B9780124167438000099.

[50] A.B. Mohamad, A.M. Zain, N.E.N. Bazin, Cuckoo search algorithm for optimization problems—A literature review and its applications, Appl. Artif. Intell. 28 (5) (2014) 419–448, http://dx.doi.org/10.1080/08839514.2014.904599.

[51] P. Mohan, R. Saranya, K. Jothi, A. Vigneshwaran, An optimal job scheduling in grid using cuckoo algorithm, Int. J. Comput. Sci. Telecommun. 3 (2012) 65–69.

[52] A. Akbarzadeh, E. Shadkam, The study of cuckoo optimization algorithm for production planning problem, ArXiv abs/1508.01310 (2015).

[53] M. Maadi, M. Javidnia, R. Ramezani, Modified cuckoo optimization algorithm (MCOA) to solve precedence constrained sequencing problem (PCSP), Appl. Intell. 48 (6) (2018) 1407–1422, http://dx.doi.org/10.1007/s10489-017-1022-0.

[54] X.-S. Yang, S. Deb, Cuckoo search: recent advances and applications. Neural computing and applications, Neural Comput. Appl. 24 (2014) http://dx.doi.org/10.1007/s00521-013-1367-1.

[55] A.H. Gandomi, X.-S. Yang, A.H. Alavi, Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems, Eng. Comput. 29 (1) (2013) 17–35.

[56] K. Chaowanawatee, A. Heednacram, Implementation of cuckoo search in RBF neural network for flood forecasting, in: 2012 Fourth International Conference on Computational Intelligence, Communication Systems and Networks, IEEE, 2012, pp. 22–26.

[57] H.R. Faragardi, M. Vahabi, H. Fotouhi, T. Nolte, T. Fahringer, An efficient placement of sinks and SDN controller nodes for optimizing the design cost of industrial IoT systems, Softw. Practice Exp. 48 (10) (2018) 1893–1919, http://dx.doi.org/10.1002/spe.2593, https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2593.

[58] Github - Optimization algorithms implementations, https://github.com/ShirinTahmasebi/Optimization-Algorithms.

[59] R. Nikouei, N. Rasouli, S. Tahmasebi, S. Zolfi, H. Faragardi, H. Fotouhi, A quantum-annealing-based approach to optimize the deployment cost of a multi-sink multi-controller WSN, Procedia Comput. Sci. 155 (2019) 250–257, http://dx.doi.org/10.1016/j.procs.2019.08.036, The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019), The 14th International Conference on Future Networks and Communications (FNC-2019), The 9th International Conference on Sustainable Energy Information Technology, http://www.sciencedirect.com/science/article/pii/S1877050919309500.

[60] F. Qian, R. Ding, Simulated annealing for the 0/1 multidimensional knapsack problem, Numer. Math. 16 (4) (2007) 320–327, http://global-sci.org/intro/article_detail/nm/8060.html.
[61] O. Titiloye, A. Crispin, Quantum annealing of the graph coloring problem, Discrete Optim. 8 (2) (2011) 376–384, http://dx.doi.org/10.1016/j.disopt.2010.12.001, http://www.sciencedirect.com/science/article/pii/S1572528610000721.
[62] R. Martonák, G. Santoro, E. Tosatti, Quantum annealing of the traveling salesman problem, Phys. Rev. E 70 (2004) 057701, http://dx.doi.org/10.1103/PhysRevE.70.057701.
[63] P. Mohan, R. Saranya, K. Jothi, A. Vigneshwaran, An optimal job scheduling in grid using cuckoo algorithm, Int. J. Comput. Sci. Telecommun. 3 (2012) 65–69.
[64] A. Akbarzadeh, E. Shadkam, The study of cuckoo optimization algorithm for production planning problem, ArXiv abs/1508.01310 (2015).
[65] M. Maadi, M. Javidnia, R. Ramezani, Modified cuckoo optimization algorithm (MCOA) to solve precedence constrained sequencing problem (PCSP), Appl. Intell. 48 (6) (2018) 1407–1422, http://dx.doi.org/10.1007/s10489-017-1022-0.

Shirin Tahmasebi received a B.Sc. in computer engineering from Shahid Beheshti University in 2017 and a M.Sc. in software engineering from Sharif University of Technology in 2020. Her main research interests include IoT, Cloud Computing, and Machine Learning.

Nayereh Rasouli received her B.Sc. in computer engineering from the university of Science and Culture and a M.Sc. software engineering in 2013 from Gazvin Islamic Azad University. Now, she works as a lecturer at university. Her research interest includes Cloud and Edge Computing, SDNs, and IoT.

Amir Hossein Kashefi received his Ph.D. from the South Tehran Branch, Islamic Azad University, Iran.

Elmira Rezabeyk received a M.Sc. from Amirkabir University of Technology.

Hamid Reza Faragardi received a B.Sc. in computer engineering in 2010 and a M.Sc. in software engineering in 2012 from the University of Tehran. He was a Ph.D. candidate in Malardalen University, Sweden in computer engineering from 2013 to 2017. Since January 2018 to March 2019 he was a postdoc researcher at the University of Innsbruck. He was also a lecturer at the KTH Royal Institute of Technology in Stockholm where he supervised multiple Master students in the context of Artificial Intelligence. Now he works as a researcher at the Ericsson Research in Stockholm, focusing on mobile edge services. He has published more than 30 peer-reviewed papers in well-known academic journals and conferences. His main research interests include Edge Computing, Cloud Computing, IoT, and Optimization Problems.