Load Balancing in SDN-Enabled WSNs Toward 6G IoE: Partial Cluster Migration Approach

Vikas Tyagi, Samayveer Singh, *Member, IEEE*, Huaming Wu, *Senior Member, IEEE*, and Sukhpal Singh Gill

***Abstract*—The vision for the sixth-generation (6G) network involves the integration of communication and sensing capabilities in Internet of Everything (IoE), toward enabling broader interconnection in the devices of distributed wireless sensor networks (WSNs). Moreover, the merging of software-defined networking (SDN) policies in 6G IoE-based WSNs i.e., SDN-enable WSN improves the network's reliability and scalability via integration of sensing and communication (ISAC). It consists of multiple controllers to deploy the control services closer to the data plane (DP) for a speedy response through control messages. However, controller placement and load balancing are the major challenges in SDN-enabled WSNs due to the dynamic nature of DP devices. To address the controller placement problem, an optimal number of controllers is identified using the articulation point method. Furthermore, a nature-inspired cheetah optimization algorithm is proposed for the efficient placement of controllers by considering the latency and synchronization overhead. Moreover, a load-sharing-based control node (CN) migration (LS-CNM) method is proposed to address the challenges of controller load balancing dynamically. The LS-CNM identifies the overloaded controller and corresponding assistant controller with low utilization. Then, a suitable CN is chosen for partial migration in accordance with the load of the assistant controller. Subsequently, LS-CNM ensures dynamic load balancing by considering threshold loads, intelligent assistant controller selection, and real-time monitoring for effective partial load migration. The proposed LS-CNM scheme is executed on the open network operating system (ONOS) controller and the whole network is simulated in the ns-3 simulator. The simulation results of the proposed LS-CNM outperform the state-of-the-art in terms of frequency of controller overload, load variation of each controller, round trip time, and average delay.**

***Index Terms*—Control node (CN) migration, controller placement problem (CPP), load balancing, multiple controllers, SDN-enabled wireless sensor network (WSN).**

I. Introduction

IN THE sixth generation (6G) network, the fusion of Internet of Everything (IoE) and wireless sensor networks (WSN) promises to revolutionize data collection, analysis, and dissemination, unlocking unparalleled potential across diverse real time applications. This revolutionary paradigm promises transformative advancements in connectivity, introducing unparalleled speeds, massive device connectivity, and seamless integration of new technologies [1]. With terabit-per-second data rates and the ability to connect a vast range of devices, 6G IoE envisions a highly integrated and interconnected network where everything from smart appliances to autonomous vehicles communicates effortlessly. A distinctive feature of 6G IoE is its commitment to sustainability, emphasizing green technologies to minimize environmental impact and ensure energy-efficient practices.

However, the convergence of the IoE with WSN in the 6G network introduces a complex and dynamic landscape of the integration of sensing and communication (ISAC) that necessitates innovative approaches to the network management and optimization [2]. ISAC stands as a pivotal advancement in IoE with WSN, bridging the gap between the efficient resource utilization and optimal performance [3]. It also addresses the demanding need for seamless coordination between the sensing and communication functions, ensuring that the sensor nodes (SNs) capture data and effectively transmit it. However, ISAC faces major challenges, such as resource constraints and potential tradeoffs between the sensing accuracy and communication efficiency [4]. In this context, the integration of software-defined networking (SDN) emerges as a pivotal solution to address the challenges and grasp the opportunities presented by this transformative paradigm. SDN enables programmability, centralized resource management, and faster policy implementation in WSN. In such an SDN-enabled WSN, SDN separates the network functions of data forwarding devices from the data plane (DP) by transferring them to a centralized controller in the control plane (CP) [5], [6]. However, when the SNs in SDN-enabled WSN exceed the threshold, the centralized controller may fail to respond to the control messages (Ctrl_Msg) from the DP devices [7].

Manuscript received 29 December 2023; revised 22 April 2024; accepted 14 May 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62071327, and in part by the Tianjin Science and Technology Planning Project under Grant 22ZYYYJC00020. *(Corresponding author: Huaming Wu.)*

Vikas Tyagi is with the Department of CSE, Dr B R Ambedkar National Institute of Technology Jalandhar, Jalandhar 144027, India, and also with the School of Computer Science Engineering and Technology, Bennett University, Greater Noida 201310, India (e-mail: itengg.vikas@gmail.com).

Samayveer Singh is with the Department of CSE, Dr B R Ambedkar National Institute of Technology Jalandhar, Jalandhar 144027, India (e-mail: samays@nitj.ac.in).

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Sukhpal Singh Gill is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, E1 4NS London, U.K. (e-mail: s.s.gill@qmul.ac.uk).

Digital Object Identifier 10.1109/JIOT.2024.3402266

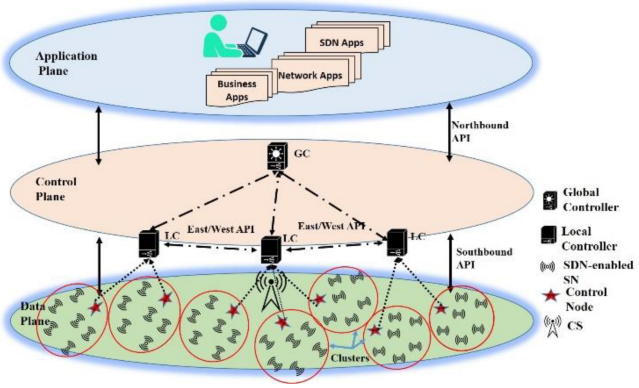

Fig. 1. Distributed SDN-enabled WSN.

Additionally, an overload situation in the SDN controller occurs when the number of Ctrl_Msg requests exceeds the maximum processing capacity of the controller. Furthermore, the SDN controller can bring down the entire network due to being a single point of failure [8], [9].

To overcome the aforementioned limitations inherent to the 6G IoE, the logical centralization of SDN-enabled WSN architecture is upgraded with the physical distribution of CP [10]. It provides a scalable and reliable distributed architecture while preserving the importance of logically centralized SDN policies as shown in Fig. 1. The local controllers (LC) are placed near the DP devices under the global controller's (GC) supervision. The communication among controllers is managed via an east-west application programming interface (API). The end user is allowed to control and manage the SDN policies in CP from the application plane through the southbound API, however, the communication between CP to DP is managed by the northbound API [11].

Distributed SDN-enabled WSN allows multiple controllers to collaborate in coordinating the network functionalities during ISAC in the 6G IoE [12]. Specifically, each controller manages clusters of SNs called control domain. However, all SNs in a cluster report to the cluster head, namely the control node (CN), and these CNs are responsible for sharing the cluster data with the corresponding controller. The allocation of clusters to each controller will be optimized to distribute network load evenly, also known as the controller placement problem (CPP) [13]. Additionally, the controller placement considers, identifying the minimum controllers and their optimal location. However, more controllers cause a high synchronization overhead in CP [14], [15].

The multiple controller architecture suffers from uneven load distribution in CP due to the dynamic nature of SDN-enabled WSN. Moreover, GC monitors each control domain periodically and migrates CN from any overloaded controller to neighboring controllers [16], [17]. However, this migration process may exceed the threshold load of the neighboring controller, leading to a change in the controller state to overload. Consequently, the migrated CN returns to the previous control domain. This phenomenon is considered as the CN Zig–Zag problem. The above issue of CN migration occurs due to CN migration as a whole. To overcome this problem, we present the load sharing-based CN migration (LS-CNM) technique, allowing the partial share of the load of an overloaded controller. LS-CNM associates an overloaded controller with an assistant controller capable of sharing the load of others. Subsequently, it selects a partial load of CNs from the overload controller domain and migrates them with the assistant controller.

This work is motivated by the need to overcome the uneven load distribution challenges in the CP of a multiple-controller architecture in the dynamic SDN-enable WSN. However, the technical challenges include the development of dynamic load balancing approaches, managing threshold load to prevent migration issues, intelligently selecting assistant controllers, designing a strategy for partial load migration, ensuring continuous load monitoring and decision making, preventing load oscillations, integrating with existing SDN infrastructure, and addressing scalability concerns. The proposed LS-CNM approach is successfully implemented to resolve the above-mentioned challenges for SDN-enabled WSNs.

To the best of our knowledge, LS-CNM is the first pioneering study that introduces dynamic management of controller workloads through the partial CN migration within distributed SDN-enabled WSNs. The main contributions of this article are summarised as follows.

1) An efficient distributed CP is devised for SDN-enabled WSN, aligning with the optimal number of controllers using the articulation point (APs) method.
2) A metaheuristic approach, referred to as CP_CO, is proposed to place the optimal number of controllers through cheetah optimization (CO), effectively addressing the CPP challenge. To refine the controller placement, a well-constructed fitness function is formulated, considering latency and synchronization overhead parameters.
3) A load sharing-based CN migration method is proposed to address the issue of load imbalance among controllers during ISAC. It examines the overloaded control domain, identifies the low-utilized assistant controller, and then chooses a suitable CN for migration based on the load of the identified assistant controller.
4) The proposed methodology is implemented on the open network operating system (ONOS) controller, and the network is simulated within the ns-3 simulator to validate its feasibility. Simulation results indicate that the LS-CNM has the capability to significantly reduce instances of controller overload while effectively achieving equitable distribution of the workload across all controllers.

The remainder of this article is organized as follows. Section II presents a summary of related work. The system model and problem formulation are presented in Section III. Section IV shows the proposed techniques. The experiment setup and simulation results of the proposed LS-CNM are discussed in Sections V. Finally, the conclusion is summarized with future directions in Section VI.

TABLE I
COMPARISON WITH OTHER RELATED WORKS

Work	Identify Optimal Controller	CPP (Metaheuristic Optimization)	Load Balancing	Controller
[14]		√		#
[16]			CR	ONOS
[12]			CM(W)	ONOS
[18]		√	FD	POX
[21]			CM(W)	Floodlight
LS-CNM	√	√	CM (W/P)	ONOS

Note: Symbol √ and # indicate adaptability and self-implemented controller, respectively. CR: Controller Reelection, FD: Flow distribution, CM(W): Cluster migration as a whole, CM(W/P): Cluster migration as a whole and partial both.

II. RELATED WORK

This section provides an overview of recent advancements in load balancing techniques for ISAC in 6G IoE-based distributed SDN-enabled WSNs, which serve as the foundation for the research background. A comparison between the previous load-balancing methods and the proposed LS-CNM scheme is discussed in Table I.

Kobo et al. [16] presented the fragmentation-based distributed control system to improve the efficiency and scalability of the software-defined WSN by bringing control services closer to the DP. It focuses on controller placement and re-election in case of failure and reduces the propagation latency. However, the controller load is not considered during controller re-election. In successive research of Kobo et al. [12], a consistent data model based on the best effort and anti-entropy strategy is considered to minimize the load during cluster switching. However, cluster switching migrates the whole cluster to another controller, overloading the controller.

Wang et al. [18] proposed a consistent load-balancing hashing algorithm using multiple controllers in underwater SDN-enabled WSNs. This approach considers an equal probability distribution process for cluster migration. However, a cluster is migrated as a whole which creates a controller Zig–Zag problem. Tahmasebi et al. [14] presented a multiobjective optimization approach for the optimal placement of SDN controllers in WSNs. This approach improves the network performance by balancing the tradeoff between the synchronization overhead and development cost. However, cluster migration is not performed for controller load balancing. Babbar et al. [19] presented two approaches for efficient cluster migration in the SDN-enabled intelligent transportation systems. The first approach detects the imbalance load among various domains, while the second approach migrates the imbalance load to another controller. However, the controller load is not managed dynamically. Whereas, this article [20] resolved this issue efficiently in SDN-enable vehicular networks by reducing cluster migration delay and cost. However, the act of cluster switching results in the complete migration of the entire cluster to another controller, leading to an overloaded state of the controller.

Salam and Bhattacharya [22] optimized CPP by minimizing both the number of controllers and network latency. This method determines the optimal number of controllers and chooses the optimal positions to place them efficiently. However, the fault tolerance approach may overload another controller in case of controller failure. Sahoo et al. [21] presented an efficient load migration technique to balance the controller load. It recognizes the under utilized controller for migration based on a selection probability. To choose the target controller, a decision analysis method ranks the under utilized controllers based on the memory, CPU load, bandwidth, and hop count. However, the cluster is migrated as a whole to another controller. Li et al. [23] optimized the CPP based on network delay and load optimization. It balances controller load by reducing network congestion and outperforms existing methods in propagation delay and load balancing in large-scale networks. However, cluster migration is not performed for load balancing.

Cheng et al. [27] presented a nested tensor-based framework that enhances ISAC using a reconfigurable intelligent surface. This structure enables joint sensing and communication without specialized pilot signals, improving detection and localization accuracy by merging the dimensions of sensing and communication signals. Li et al. [28] explored physical–layer authentication (PLA) for the user identification and security in the AmBC-based NOMA symbiotic networks, taking into account channel estimation errors when assessing false alarms and detection probabilities for distant and nearby users. Gill et al. [29] introduced a classification framework for modern computing based on performance and impact, categorizing it by paradigms, technologies, and trends. Montazerolghaem [30] discussed a method that managing resources optimally in the Internet of Medical Things (IoMT) networks, considering both the energy and load constraints. Then, the author introduced a system that manages energy and load in IoMT by leveraging network softwarization and virtual resources. This system dynamically modifies resource allocations based on the real-time size of the IoMT network. Montazerolghaem and Yaghmaee [31] introduced a new framework that utilizes SDN to meet the QoS demands of diverse IoT services while also managing traffic distribution among IoT servers. The authors suggest a forward-looking heuristic approach, which integrates time-series analysis and fuzzy logic to predict and manage network conditions. Montazerolghaem introduced a framework for data centers utilizing SDN to evenly distribute server loads and prevent server overloads [32]. The framework also delivers services quickly with minimal computational complexity. Alhilali and Montazerolghaem [33] discussed an SDN architecture and explored load balancing challenges within it. They also categorize artificial intelligence (AI)-based load balancing methods, evaluating them based on the algorithms used, the problems addressed, and their pros and cons.

III. SYSTEM MODEL AND PROBLEM FORMULATION

In this section, the characteristics of a multicontroller-based SDN-enabled WSN model are introduced for ISAC among the network devices. Then, the CPP and CN migration problems are formulated.

TABLE II
SYMBOLS AND EXPLANATION

Symbols	Explanation
$\hat{Q}$ and $\hat{C}$	Set of CNs and Controller respectively
c_i, and cn_i	The i^{th} controller and control node, respectively
$\hat{Q}_{ci}$	Set of CNs which are managed by the c_i
$\hat{C}_{OL}$ and $\hat{C}_{Ast}$	Set of overloaded and assistant Controller
$Lat^{Avg}cn, c(P)$	Average latency between CN and Controller
$Lat^{Avg}c, c(P)$	Average latency between Controller to Controller
$Lat^{Avg}c, GC(P)$	Average latency between Controller to Global Controller
$Syn\_d_{c_i,c_j}$	Synchronization delay between c_i and c_j
$Crt\_L^t(c_i)$	Current load of i^{th} controller in time period t
$CM\_cn_j^t$	Control messages sent by j^{th} CN in time period t
Φ	Threshold value of controller load
$X^t_{CH_i,j}$, $X^t_{P_i,j}$	Position of cheetah and prey in dimension j
$S^t_{CH_i,j}$	The step size of cheetah
$\check{\Upsilon}_{CH_i,j}$, $\check{\Upsilon}^t_{CH_i,j}$	Turning factor and Interaction factors of cheetah

A. Characteristics of Proposed Network Model

The proposed 6G IoE-based SDN-enabled WSN model is considered as an undirected graph $G = (V, E)$, where V represents the set of CNs and controllers, and E represents the set of links between the CNs and controllers as shown in Fig. 1. Let $\widehat{Q} = \{cn_1, cn_2, \ldots, cn_n\}$ and $\widehat{C} = \{c_1, c_2, \ldots, c_m\}$ are the set of n CNs and m controllers, respectively, where $\widehat{Q}$, $\widehat{C} \in V$. However, the CPP is an optimization problem, which focuses on finding the optimal controller positions among a large number of potential options. The following list of presumptions pertains to dynamic controller placement based on latency and load balancing.

1) The SNs are deployed randomly and CS is placed at the centre of the target-sensing region in the DP.
2) All the devices, participating in ISAC are stationary in the network scenario and the network load is dynamic in nature.
3) GC is connected with DP using the LCs and all SNs are capable of performing the responsibilities of a CN.
4) Each c_i is capable of acting as the master controller of any CN where each c_i can respond to requests of one or more CNs in accordance with its processing capacity.
5) The proposed method enables the clusters to migrate partially/completely with another c_i to distribute the load evenly. Each control domain is assigned one c_i and multiple CNs.

The symbols used in this article with their explanation are presented in Table II.

B. Controller Placement Problem

The CPP is optimized by determining the optimal controllers and their locations using the minimal controllers, latency, and synchronization overhead. It balances the network load that ensures efficient communication among SNs and controllers in the ISAC process.

1) Optimal Number of Controllers: The networks equipped with more controllers, decrease the overall latency but increase the communication overhead between the controllers. Therefore, it is essential to determine the optimal number of controllers. The optimal number of controllers is called m, i.e., elected using Algorithm 1, based on AP to balance tradeoff between the latency and communication overhead.

Algorithm 1: Optimal Number of Controllers Module

Input: Network Graph $G = (V, E)$
Output: Number of articulation points (Controllers)

```
Initially all vertices ← not visited
Create function
  Art_Point (vert, N_visited[ ], N_parent[ ], Art_P[ ])
Call the function Art_Point, recursively
Child_node ← 0
N_visited[u] ← Set True
Visit all the vertices adjacent to N_visited[ u] // Calculate
  the depth of the selected vertex
if N_visited[v] is not True
    Child_node ← Child_node + 1
    N_parent[v] ← Set u
    if (subtree has any connection with any of the
     ancestors is True)
        no articulation points // u is root of DFS tree and
         has two or more children
    else if
        (N_parent [u]==NILL and Child_node>1)
         Art_P[u] ← Set True
    End
End
Else
    Call the function Art_Point
End
return Art_P[ ]
```

2) Latency: The latency between a CN and its respective controller is the average distance that a data packet (P) travels from the cn_n to c_m. It is represented by $\text{Lat}^{\text{Avg}}cn, c(P)$ as given in

$$\text{Lat}^{\text{Avg}}cn, c(P) = \frac{1}{n} \sum_{cn \,\epsilon\, \widehat{Q}} \min D(cn, c). \tag{1}$$

The intercontroller latency is the average distance that a packet travels from one controller to another (local or global). It is represented as $\text{Lat}^{\text{Avg}}c, c(P)$ and $\text{Lat}^{\text{Avg}}c, GC(P)$ as given in (2) and (3) for LC to LC and LC to GC, respectively.

$$\text{Lat}^{\text{Avg}}c, c(P) = \frac{1}{m} \sum_{i,j=0}^{m} \min_{c \in \widehat{c}} D\left(c_i, c_j\right) \tag{2}$$

$$\text{Lat}^{\text{Avg}}c, GC(P) = \frac{1}{m} \sum_{i=0}^{m} \min_{c \in \widehat{c}} D(c_i, GC). \tag{3}$$

The total latency (AVG_Lat(P)) is the sum of $\text{Lat}^{\text{Avg}}cn, c(P)$, $\text{Lat}^{\text{Avg}}c, c(P)$ and $\text{Lat}^{\text{Avg}}c, GC(P)$ latencies as given in

$$\begin{aligned}\text{AVG\_Lat}(P) \;&= \text{Lat}^{\text{Avg}}cn, c(P) \\ &\quad + \text{Lat}^{\text{Avg}}c, c(P) + \text{Lat}^{\text{Avg}}c, GC(P).\end{aligned} \tag{4}$$

3) Synchronization Overhead: The synchronization overhead represents the additional communication required to

coordinate with multiple controllers. It includes tasks, such as exchanging status updates, coordinating actions, and resolving conflicts. The extent of synchronization overhead depends on the specific system and the complexity of the controllers. To measure the synchronization overhead between each pair of controllers (c_i, c_j), a matrix M_Syn is defined as the number of synchronization messages exchanged between c_i and c_j. Thus, the synchronization overhead is denoted by Syn_o, and formulated as follows:

$$Syn\_o = \sum_{c_i \in C} \sum_{c_j \in C} Syn\_d_{c_i,c_j} * M\_Syn_{c_i,c_j} \quad (5)$$

where $Syn\_d_{c_i,c_j}$ and $M\_Syn_{c_i,c_j}$ represent the synchronization delay and messages between c_i and c_j, respectively.

C. Load Balancing

In a multicontroller SDN-enabled WSN, the network load balancing involves the systematic distribution of traffic among multiple controllers. This strategic approach aims to optimize resource utilization and enhance overall network performance in ISAC approach among network devices. This is achieved by considering both the capacity of the controllers and the migration of CNs.

1) Controller Capacity: It refers to the highest number of requests that a controller can handle at a specific time period t. The maximum capacity of a controller indicates how many $Ctrl\_Msg$ can be processed in t, i.e., represented as $Max\_L(c)$. All LCs have a similar capacity and the current load $Crt_L{}^t(c_i)$ of c_i at time t is given as follows:

$$Crt\_L^t(c_i) = \sum_{j=0}^{k} CM\_cn_j^t \quad (6)$$

where $CM\_cn_j^t$ represents the (Ctrl_Msg) sent by CN that exist in the control domain of c_i. When the current load is exceeded to $Max\_L(c)$, the performance of any controller may degrade, and initiate the cluster/CN migration to maintain the stability of the network.

2) Cluster/CN Migration: The process of moving a CN from one controller domain to another to balance the load of an overloaded controller is called CN migration. CN migration is triggered by various factors, such as network congestion, changes in traffic pattern, and network failures. The decision to migrate a CN to a particular controller is based on the current load of the neighboring controller. Additionally, the neighboring controller immediately eliminates the migrated CN if its $Crt_L{}^t(c_i)$ is exceeded due to the migrated CN. Subsequently, the CN returns its original domain and initiates another CN migration process due to the overloaded state of the controller. This situation gives rise to the CN migration problem, which occurs as a consequence of CN migration as a whole.

An example is illustrated in Fig. 3(a), which shows a scenario of the CN migration problem and its solution. Assume Φ is the threshold load of the c_i where $\Phi < Max\_L(c)$ and controller c_i is considered as overloaded if $Crt_L{}^t(c_i) > \Phi$. In Fig. 3(a), there are two controllers c_1 and c_2 with $\Phi_1 = \Phi_2 = 70$ and three CN, namely cn_1, cn_2 and cn_3 in a network.

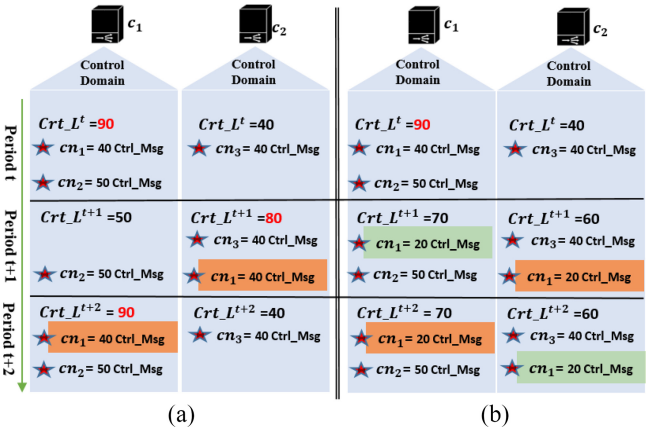

Fig. 2. Illustrate CN Zig–Zag problem and how LS-CNM solves it. (a) CN migration problem. (b) Load sharing-based CN migration solution.

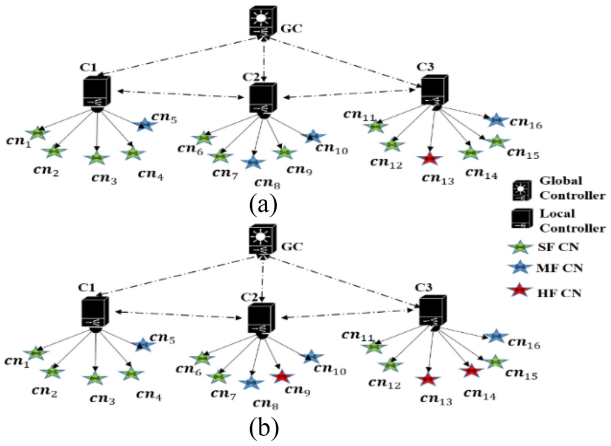

Fig. 3. Control Domains at (a) 1 s. (b) 10 s.

Moreover, cn_1, cn_2 and cn_3 produce 40, 50 and 40 $Ctrl\_Msg$, respectively, in period t. Controller c_1 takes the charges of cn_1 & cn_2 and cn_3 is controlled by c_2. Now, controller c_1 is overloaded because $Crt_L{}^t$ (c_1) is 90 $Ctrl\_Msg$, i.e., greater than Φ, and thus CN migration is required. In the current state of the load balancing mechanism [12], an overloaded controller c_1 asks c_2 to take responsibility for some of its CNs as a whole for an entire period t as shown in Fig. 3(a). Furthermore, cn_1 migrates to the control domain of c_2 at $t+1$. Since, the $Crt_L{}^t(c_2) >$ 70 due to newly migrated CN. Now, c_2 asks c_1 to take charge of cn_1 for period $t+2$. Accordingly, the current situation at c_1 is the same as period t, and this is called the CN Zig–Zag problem.

Besides the scenario mentioned above, the proposed LS-CNM performs CN migration in a partial load sharing manner for specific period t. During this period, the load of a CN is split between the two controllers, as shown in Fig. 3(b). At period $t+1$, the load of cn_1 is shared between c_1 and c_2 to ensure that the load remains below the threshold i.e., $Crt_L{}^{t+1}(c_i) < \Phi$. This approach helps in keeping the workloads of both the controllers below their thresholds. In this way, LS-CNM can effectively address the issue of the CN Zig–Zag problem during migration.

IV. Proposed Methodology

In this section, CPP is optimized in accordance with an optimal number of controllers and their best location in the SDN-enabled WSN during ISAC among the 6G IoE devices. After that the CN Zig–Zag problem is resolved using a load sharing-based partial CN migration technique.

A. Controller Placement

The objective of the controller placement phase is to determine the necessary quantity of controllers and the position of each controller at an optimized location to maintain the network stability and efficiency.

1) Optimal Number of Controllers: A method from the graph theory is employed to calculate the optimum numbers of controllers and identify initial controller locations within the given network topology by identifying APs [24]. An AP is defined as a vertex/node whose removal may result in the partitioning of the graph. The value of identified APs is used as the required optimal number of the controller in the proposed network. The conventional depth-first search (DFS) [25] algorithm is employed to identify the APs within the network. In Algorithm 1, a vertex or node "u" is considered as the parent of another vertex "v" if and only if "v" can be discovered by traversing from "u". A vertex "u" is classified as an AP if any of the following criteria are met.

1) Vertex "u" is the root node and has a minimum of two child nodes.
2) Vertex "u" is not the root node and has a child "v", where there is no path of connectivity between "v" and any of the ancestors of "u" in the DFS tree.

In Algorithm 1, the next visited node is designated as "u_i''", and a data structure N_{visited} is utilized to record the nodes that have been traversed in the graph. The algorithm progresses by traversing all the neighboring nodes of the currently visited node. At each iteration, the values of the visited nodes are updated. If a neighboring node has not been visited, it is considered as a $\text{Child}_{\text{node}}$ of the current node, and its connectivity to any ancestors is evaluated. If there is no connectivity, the node is classified as an AP.

2) Controller Placement Based on Cheetah Optimization: Once the necessary quantity of controllers has been identified, synchronization-aware controller placement in SDN-enabled WSNs is performed by utilizing CO as outlined in Algorithm 2. CO motivates the selection of the best prey from multiple prey as CNs for each cheetah acting as controllers. A cheetah's decision on the best prey to pursue is represented by a fitness function and different prey options constitute the potential solutions. This optimization is based on the cheetah's hunting strategies, such as searching, sitting-and-waiting, attacking, leaving the prey, and going back home defined as follows.

Searching Strategy: The cheetahs' searching strategy is mathematically modeled using the variable $X^t_{CH_i,j}$ which represents the current position of the cheetah $CH_i (i = 1, 2, \ldots, n)$ in search space dimension ($j = 1,\ 2,\ \ldots,\ D$), where n is the number of cheetahs in the population and D is the dimension of the optimization problem. Each cheetah reaches at different positions when hunting various prey. Using this information, a random search (7) is utilized to find the new position $X^{t+1}_{CH_i,j}$ based on their current position and an arbitrary step size

$$X^{t+1}_{CH_i,j} = X^t_{CH_i,j} + r^{-1}_{CH_i,j} \cdot S^t_{CH_i,j} \tag{7}$$

where $r_{CH_i,j}$ represents the random number generated using the normal distribution method. $S^t_{CH_i,j}$ represents the step size of cheetah in hunt time t. $S^t_{CH_i,j}$ is calculated as $S^t_{CH_i,j} = 0.001 \times t/T$, where T represents the maximum allowed hunting duration i.e., calculated as $T \leftarrow 60 \times \lceil D/10 \rceil$.

Sitting-and-Waiting Strategy: The cheetah chooses to sit-and-wait, in order to get close enough to the prey. In this mode, the cheetah remains in its current position and waits for

Algorithm 2: *CP_CO*

Input: Initialize the position of GC, CS and CNs (Prey), dimension (D), Initial population size (P_s)
Output: Best position for each controller

Generate the initial position of search agent $X^t_{CH_i,j}$ $(i = 1,\ 2, \ldots, n)$ and $(j = 1,\ 2, \ldots, D)$
Evaluate the fitness of each search agent CH_i using (12)
Initialize the population's home, leader, and prey solutions
$t \leftarrow 0$, $IT \leftarrow 1$, $IT_{Max} \leftarrow$ Set as Maximum Iterations
Calculate $T \leftarrow 60 \times \lceil D/10 \rceil$
while *current itertaion* $IT \leq IT_{Max}$ **do**
  Select random search agent $CH(2 \leq Ch \leq n)$
  for *each search agent* $i \in m$ **do**
    Define neighbor search agents' set of CH_i
    for *each arbitrary arrangement* $j \in \{1,\ 2, \ldots, D\}$ **do**
      Calculate H, $r_{CH_i,j}$, $\check{T}_{CH_i,j}$, $S^t_{CH_i,j}$, $\check{I}^t_{CH_i,j}$, and choose random numbers Rnd_1, Rnd_2 and Rnd_3 uniformly from 0 *to* 1
      if $(Rnd_2 < Rnd_3)$ **then**
        Choose random number Rnd_4 from 0 *to* 3
        if $(H \geq Rnd_4)$ **then**
          Update new position of search agent using (7) // *Searching mode*
        Else
          Update new position of search agent using (9) // *Attacking mode*
        End
      Else
        Update new position of search agent using (8) // *Sit-and-wait mode*
      End
    End
    Update the solutions of search agent i and the leader
  End
  $t \leftarrow t + 1$
  if $t > Rnd_2 \times T$ **then**
    $X^t_{CH_i,j} \leftarrow X^{t-1}_{CH_i,j}$ the leader position doesn't change // *Leave the prey and go back home mode*
    Evaluate the fitness of each search agent CH_i
    $t \leftarrow 0$
  End
  $IT \leftarrow IT + 1$
  Update the global best for leader search agent
end
if $(i < n)$ **then**
  Exclude the current leader search agent and go to step 3
Else
  Update the global best for each search agent
End

the prey to come within reach. This behavior is represented as follows.

$$X^{t+1}_{CH_i,j} = X^{t}_{CH_i,j} . \quad (8)$$

This approach involves gradually changing the cheetahs in each group rather than all at once, which improves the chances of finding a better solution and prevents the algorithm from reaching a suboptimal solution too quickly.

Attacking Strategy: When a cheetah chooses to hunt, it uses two critical elements: 1) speed and 2) flexibility. The cheetah rushes toward its prey at top speed. The cheetah tracks the position of its prey and alters its path to intercept the prey's path at a specific point. The position of the cheetah will be updated as follows.

$$X^{t+1}_{CH_i,j} = X^{t}_{P_i,j} + \check{\mathrm{T}}_{CH_i,j} . \check{\mathrm{I}}^{t}_{CH_i,j} \quad (9)$$

where $X^{t}_{P_i,j}$, $\check{\mathrm{T}}_{CH_i,j}$ and $\check{\mathrm{I}}^{t}_{CH_i,j}$ represent the prey location, turning factor, and interaction factor associated with cheetah, respectively. $\check{I}^{t}_{CH_i,j}$ is used to prevent collision during the attack and denoted as the difference between the cheetah's current position $X^{t}_{CH_i,j}$ with neighboring group of cheetahs' $X^{t}_{CH_k,j}$, where $k \neq i$. The turning factor $\check{\mathrm{I}}^{t}_{CH_i,j}$ shows the sudden turn of CH_i, j while hunting and it can be formulated as $\check{\mathrm{I}}^{t}_{CH_i,j} = |r_{CH_i,j}|^{\exp((r_{CH_i,j})/2)}.\sin(2\pi.r_{CH_i,j})$. During hunting period, cheetah switches between the searching, sit-and-wait, and attacking mode as per the rules expressed in

$$\begin{cases} \text{if } (\mathrm{Rnd}_2 \geq \mathrm{Rnd}_3), \text{ Sit and Wait} \\ \text{if } (\mathrm{Rnd}_2 < \mathrm{Rnd}_3),\ H = e^{2(1-t/T)}\ (2\mathrm{Rnd}_1 - 1) \end{cases} \quad (10)$$

$$\begin{cases} \text{if } (H \geq \mathrm{Rnd}_4), \text{ Attack Mode} \\ \text{if } (H < \mathrm{Rnd}_4), \text{ Searching Mode} \end{cases} \quad (11)$$

where Rnd_1, Rnd_2, and Rnd_3 are random numbers in the range of [0, 1]. H is a switching factor and Rnd_4 is a random value in the range of [0, 3]. If CH_i fails multiple hunts, their position is replaced by the last successfully hunted prey location, this strategy is called leave the prey and go back home mode.

The CP_CO algorithm is used to determine the optimal location of controllers for controller placement in the CP. In the proposed work, the number of CNs and their position are generated for the clusters similar to those defined in GMPSO [26]. After that, each $cn_j \in \widehat{Q}$ selects their master controller $c_i \in \widehat{C}$ based on the latency factor as in (1). This process creates $\widehat{Q}_{ci}$ as the set of CNs i.e., managed by c_i. Moreover, $\widehat{Q}_{ci}$ is updated after each reclustering process.

Fitness Function: The CP_CO is employed to find solutions quickly i.e., close to optimal during the controller placement. The latency and synchronization overhead are integrated into a single fitness function f_{Fit} as in (12). This allows to identify efficient solutions that are near the global optimum while ensuring that the optimal controller placement constraints are not violated

$$f_{\mathrm{Fit}} = \alpha.\mathrm{AVG}_{\mathrm{Lat}}(P) + \beta.Syn\_o \quad (12)$$

where $\alpha$and β are tuning constant values and considered as $\alpha + \beta = 1$. These values are used to tune the relative significance of the $\mathrm{AVG}_{\mathrm{Lat}}(P)$ and Syn_o in the network.

Algorithm 3: LS-CNM

Input: $\widehat{Q}$ $\widehat{C}$, $\widehat{Q}_{ci}$, $Max\_L(c)$, Φ,
Output: Balanced control node migration

Initially $\widehat{C}_{OL}$ *and* $\widehat{C}_{Ast} \leftarrow \{\ \}$
for *each* $c_i \in \widehat{C}$ **do**
  $Crt\_L^t(c_i) \leftarrow 0$
  for *each* $cn_j \in \widehat{Q}$ **do**
    $Crt\_L^t(c_i) = Crt_L{}^t(c_i) + CM\_cn_j^t$
  End
  if $(Crt_L{}^t(c_i) > \Phi)$ **then**
    $\widehat{C}_{OL} \leftarrow \widehat{C}_{OL} \cup \{c_i\}$
  else
    $\widehat{C}_{Ast} \leftarrow \widehat{C}_{Ast} \cup \{c_i\}$
  End
End
if *(*$\widehat{C}_{OL}$ *and* $\widehat{C}_{Ast}$is not empty*)* **then**
  SORT$(\widehat{C}_{OL},\ Crt_L{}^t(c_i) - \Phi)$
  SORT$(\widehat{C}_{Ast}, \Phi - Crt_L^t(c_i))$
End
Terminate the process of *LS-CNM*
for *each* c_i $\widehat{C}_{OL}$ **do**
  SORT$(\widehat{Q}_{ci},\ CM_cn_i^t\ )$
  while *each* $Crt_L{}^t(c_i) > \Phi$ **do**
    Choose the nearest controller $c_j \in \widehat{C}_{Ast}$ *// based on latency (2) synchronization overhead (5)*
    Migrate $cn_i \in \widehat{Q}_{ci}$ with c_j subnet till $Crt_L{}^{t+1}(c_i) > Crt_L{}^t(c_i) + CM\_cn_i^t$
    Calculate $PS\_CM\_cn_i^t = \Phi - Crt_L{}^t(c_i)$
    $Crt_L{}^t(c_i) \leftarrow Crt_L{}^t(c_i) - PS_{CM_{cn_i^t}}$
    $Crt_L{}^t(c_j) \leftarrow Crt_L{}^t(c_j) + PS_{CM_{cn_i^t}}$
  end
  if $(Crt_L{}^t(c_j) > \Phi)$ **then**
    $\widehat{C}_{Ast} \leftarrow \widehat{C}_{Ast} \setminus \{c_j\}$
  else
    SORT$(\widehat{C}_{Ast}, \Phi - Crt_L{}^t(c_j)\ )$
  End
  if $(\widehat{C}_{Ast} \leftarrow \{\ \})$ **then**
    Terminate the process of *LS-CNM*
  End
End

B. Load Balancing

At the primary stage of the network, each CN chooses one controller as a master controller and creates an initial subnet. The load-sharing-based CN migration scheme defined in Algorithm 3 allows CNs to migrate with controllers using partial load sharing rather than as a whole CN. It also allows more flexibility and addresses the issue of CN Zig–Zag during the migration process.

LS-CNM begins by identifying $\widehat{C}_{OL}$ and $\widehat{C}_{\mathrm{Ast}}$ as the set of overloaded and assistant controllers from $\widehat{C}$. According to (6), step 5 calculates the current load of each controller $c_i \in \widehat{C}$. If the current load exceeds a threshold Φ, c_i is classified as overloaded else classified as an assistant controller. Then, steps

13 to 17 show if both the sets $\widehat{C}_{OL}$ and $\widehat{C}_{\text{Ast}}$ are not empty, the migration process is initiated for load-balanced. The migration starts by sorting the $\widehat{C}_{OL}$ and $\widehat{C}_{\text{Ast}}$ in decreasing order of overload controllers and the remaining controllers' capacity, respectively. Then, it iteratively selects a pair of controllers, one overloaded and other assistants, and then migrates one CN from the overloaded controller's subnet to the assistant's in order to reduce the overloaded controller's current load. In this process, step 23 identified the number of partial sharing (Ctrl_Msg) ($PS\_CM\_cn_i^t$) of migrating CN in accordance with the $Crt_L{}^t(c_j)$ will not be exceeded from Φ. This process continues until either the set of overloaded controllers or the set of assistant controllers is empty. The module stops if there are no more assistant controllers available to share the current load of the overloaded controllers.

C. Complexity Analysis

Algorithm 1 aims to determine the optimal number of controllers in a network by identifying APs. The initialization process involves marking all vertices as not visited, which takes $O(v)$ time, where v is the number of vertices in the graph. Afterward, DFS traversal is performed to visit all vertices and calculate the depth of the selected vertex. The time complexity of a standard DFS is $O(v + E)$, where E is the number of edges in the graph. The recursive calls to the Art_Point function occur for unvisited vertices. In the worst case, each vertex is visited once, leading to a total time complexity of $O(v)$. Considering the above components, the overall time complexity of the algorithm is dominated by the DFS traversal and can be expressed as $O(v + E)$. The space complexity is influenced by the stack space used in the recursive calls and can be expressed as $O(v)$.

The proposed CP_CO algorithm, as outlined in Algorithm 2, is designed for optimizing controller positions. Initially, the process begins with the initialization phase, which includes generating the initial positions of the search agents. The time complexity of the initialization phase is $O(P_s * D)$, where P_s is the initial population (IP) size, and D is the dimension. Afterward, the fitness evaluation of each search agent has a time complexity of $O(P_s)$. The main loop iterates for a maximum of IT_{Max} iterations. The loop involves operations, such as selecting random search agents, defining neighbor sets, and updating agent positions. By considering these components, the overall time complexity of the CP_CO algorithm is influenced by the main loop, nested loops, and update operations. Therefore, the overall time complexity is approximately $O(\text{IT}_{\text{Max}} * P_s * D)$. The space complexity is determined by the storage of search agent positions and additional variables and can be expressed as $O(P_s * D)$.

The proposed LS-CNM algorithm, as outlined in Algorithm 3, is designed for CN migration in a network. The initialization section involves creating two sets, $\widehat{C}_{OL}$ and $\widehat{C}_{\text{Ast}}$, and initializing some counters. This part has a time complexity of $O(|\widehat{C}|)$, where $|\widehat{C}|$ is the size of the set $\widehat{C}$. The first loop iterates through each CN in $\widehat{C}$. Inside the loop, there is a nested loop that iterates through each controller in $\widehat{C}_{\text{Ast}}$. The operations inside the nested loop have a time complexity of $O(|\widehat{C}_{\text{Ast}}|)$. Overall, the time complexity of the first loop is $O(|\widehat{C}|) * |\widehat{C}_{\text{Ast}}|)$. Subsequently, the SORT operations for sets $\widehat{C}_{OL}$ and $\widehat{C}_{\text{Ast}}$ have time complexities of a standard sorting algorithm i.e., is typically $O(n \log n)$, where n is the size of the set being sorted. Therefore, the time complexity of the sorting operations is $O(|\widehat{C}_{OL}| * \log(|\widehat{C}_{OL}|))$ and $O(|\widehat{C}_{\text{Ast}}| * \log(|\widehat{C}_{\text{Ast}}|))$. Similarly, the time complexity of the second loop is determined by the operations inside the while loop, and it depends on the specific input and conditions. In the worst case, it may be $O(|\widehat{C}_{OL}| * |\widehat{Q}_{ci}|)$. Finally, the time complexity of LS-CNM is primarily influenced by the sizes of the sets $\widehat{C}$ $\widehat{C}_{\text{OL}}$, $\widehat{C}_{\text{Ast}}$, and $\widehat{Q}_{ci}$ and the sorting operations within the algorithm. Accordingly, the overall time complexity is dominated by the sorting operations, and it can be expressed as $O(|\widehat{C}|+|\widehat{C}_{OL}|*\log(|\widehat{C}_{OL}|)) + |\widehat{C}_{\text{Ast}}|* \log(|\widehat{C}_{\text{Ast}}|)+ |\widehat{Q}_{ci}|* \log(|\widehat{Q}_{ci}|)$). The space complexity of LS-CNM is influenced by the sizes of the sets $\widehat{C}$, $\widehat{C}_{\text{OL}}$, $\widehat{C}_{\text{Ast}}$, and $\widehat{Q}_{ci}$, as well as the temporary variables used in the algorithm. Thus, the space complexity can be expressed as $O(\widehat{C} + \widehat{C}_{\text{OL}} + \widehat{C}_{\text{Ast}} + \widehat{Q}_{ci})$.

In conclusion, it is essential to aggregate the complexities of all algorithms to calculate the overall time and space complexity of the proposed work. Thus, the overall time complexity is $O(v + E + \text{IT}_{\text{Max}} * P_s * D + |\widehat{C}| + |\widehat{C}_{OL}| * \log(|\widehat{C}_{OL}|)) + |\widehat{C}_{\text{Ast}}| * \log(|\widehat{C}_{\text{Ast}}|) + |\widehat{Q}_{ci}| * \log(|\widehat{Q}_{ci}|))$ and the overall space complexity can be expressed as $O(v + P_s * D + \widehat{C} + \widehat{C}_{\text{OL}} + \widehat{C}_{\text{Ast}} + \widehat{Q}_{ci})$.

V. Performance Evaluation

This section provides the detailed result and discussion obtained from the proposed *LS-CNM,* Kobo et al. [12], and the OpenFlow protocol. The performance of *LS-CNM,* Kobo et al. [12], and the OpenFlow protocol are analysed using various network performance metrics like frequency of controller overload (FCO), load on controllers, round trip time (RTT), and average delay.

A. Experimental Setup

The proposed approach is implemented on the ONOS controller (Junco ver-1.9.2) and the network is simulated within the ns-3 network simulator ver-3.26. These tools are installed on Ubuntu OS (16.04-LTS) with an Intel i7 10th generation processor and 16 GB of RAM. The OpenFlow version 1.3 is used as the southbound interface to connect ONOS and ns-3. The network simulator parameters are presented in Table III.

An instance of the proposed network topology used in our implementation is depicted in Fig. 4, which consists of 4 controllers $\widehat{C} = \{c_0, c_1, c_2, c_3\}$ and 16 CNs $\widehat{Q} = \{cn_1, cn_2, cn_{16}, \}$, however, the CNs are dynamic. The hierarchical paradigm of the distributed architecture is adopted, where controller c_0 serves as the GC. GC coordinates all LC as well as monitors the load of each one to determine CN migration.

However, c_0 does not participate in CN migration. Each controller, excluding c_0, has a maximum processing capacity ($Max\_L(c)$) of 100*Ctrl_Msg* per second and a threshold (Φ) is 70% of $\text{Max}_{L(c)}$. If any LC receives more than 70% *Ctrl_Msg*

TABLE III
SIMULATION PARAMETERS

Parameters	Values
Size of Network	$200 \times 200\ m^2$
Total Nodes in the Network	300
CS Location	(100*100)
SNs Initial energy	1J
Packet size	2000 bits
Transmitter and receiver energy consumption	50nJ/bit
Simulation Time	280 Sec
Time period t	5 Sec

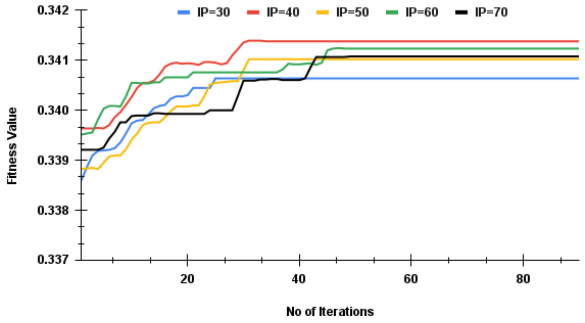

Fig. 4. Convergence of CP_CO through IP, fitness, and iterations.

in a period (t), the respective controller is considered as overloaded.

In addition, three types of CNs are considered based on the frequency (like small, medium, and higher) of *Ctrl_Msg* generation as shown in Fig. 4. Each CN with small frequency (SF), medium frequency (MF), and high frequency (HF) generates 8 to 10, 10 to 13, and 13 to 16 *Ctrl_Msg* per second, respectively. Fig. 4 shows a simulation instance of the control domain for each LC at 1 sec, where all CNs except for cn_5, cn_8, cn_{10}, cn_{13}, and cn_{16}, are SF CNs. After 10 sec, cn_9 and cn_{14} become CNs of HF which can cause CN migration as depicted in Fig. 4(b).

Moreover, the experimentation is also considered by varying the IP and the number of iterations within the ranges of 30 to 70 and 5 to 90, respectively, to observe how different combinations of population sizes and iterations would impact the convergence of the CP_CO algorithm. Based on the analysis, it is found that the best population size (IP $=$ 30) led to convergence after approximately 25 iterations based on the fitness values in each iteration, as illustrated in Fig. 5. This indicates that, among the tested various population sizes, an IP of 30 individuals demonstrated optimal convergence behavior for the proposed CP_CO algorithm. The next section compares the performance of the proposed *LS-CNM* with Kobo et al. [12] and the OpenFlow protocol.

B. Frequency of Controller Overload

The FCO determines how many times a controller exceeds their threshold capacity Φ. Moreover, numerous simulations are conducted to enhance result realism. The depicted average in Fig. 6 is accompanied by a 95% confidence interval to provide a measure of result reliability. Fig. 6 depicts the frequency of the controller overloaded with respect to each controller.

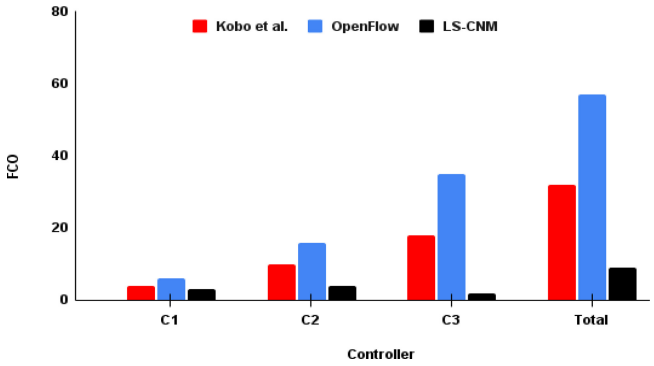

Fig. 5. FCO.

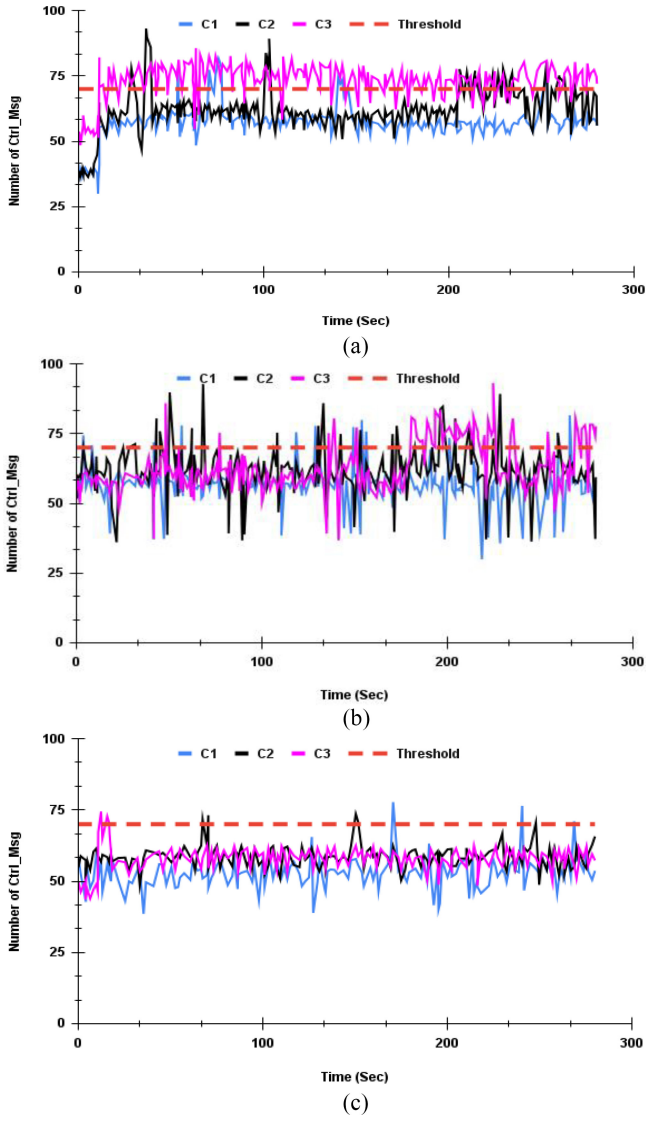

Fig. 6. Load evaluation of each controller (a) OpenFlow. (b) Kobo et al. [12] (C) LS-CNM.

It is evident from Fig. 5 that LS-CNM reports less FCO in comparison with Kobo et al. [12] and OpenFlow. The LS-CNM not only selects the most appropriate CN for migration but also elects partial load during migration. Moreover, it allows neighboring controllers to handle the process of the

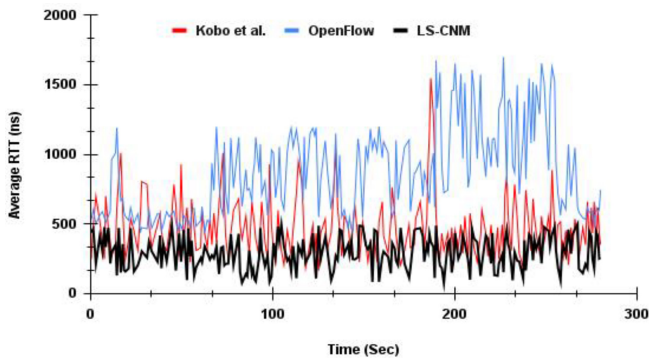

Fig. 7. Average RTT.

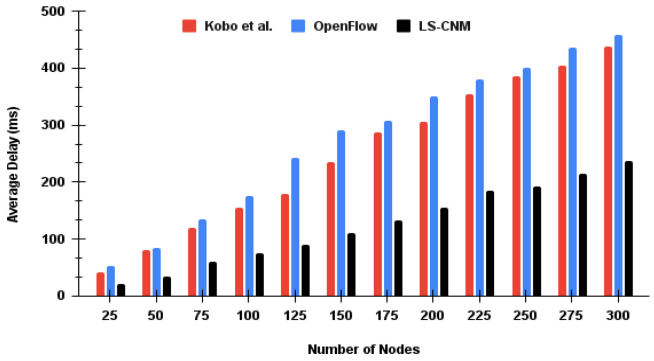

Fig. 8. Average delay versus number of nodes.

overloaded controller during the same time period. This leads to a further decrease in the FCO of all controllers as compared to Kobo et al. [12] and OpenFlow.

C. Load on Controllers

Load on the controller represents how many *Ctrl_Msg* are processed per unit of time. Fig. 7 shows a comparative analysis of the number of *Ctrl_Msg* received by each controller with respect to time for proposed *LS-CNM,* Kobo et al. [12] and the OpenFlow protocol. In Fig. 7, the values of some points are greater than the threshold, indicating that the current load of the specific controller has surpassed the threshold, leading to a situation of controller overloading in the network.

In Fig. 7(a), the simulation result of OpenFlow shows that the controller c_3 remains in the overloaded state for a long period because neighboring controllers also reached their thresholds frequently. In Fig. 7(b), Kobo et al. [12] show the long period of overloaded states of controllers when c_2 and c_3 are overloaded simultaneously due to the absence of load balancing.

Fig. 7(c) depicts that the instance at 11 s when the controller $C3$ is identified as overloaded and controller $C3$ changes its state to normal within 5 s due to the partial load-sharing migration in LS-CNM. It enables two controllers to jointly handle the processing of *Ctrl_Msg* of HF of CN which leads to a better distribution of workloads. During the simulation period, the load of each controller becomes very similar to each other which ensures LS-CNM can effectively balance the load among all controllers.

D. Average Round Trip Time

It refers to the average time taken by a *Ctrl_Msg* to be processed from end to end. In addition, the average time in which the *Ctrl_Msg* to be sent from the CN to the controller, is processed by the respective controller, and then returned to the CN. Fig. 8 depicts the RTT of *Ctrl_Msg* transmitted over time. A high RTT can result in significant delays in the processing of packets, leading to slow network performance. The efficient controller placement based on the latency and synchronization overhead in LS-CNM reports less RTT in Fig. 8 compared with Kobo et al. [12] and OpenFlow.

E. Average Delay

It refers to the average time taken for a data packet to be transmitted from an SN to a control server. This delay includes the time required for the data to be processed at SN, transmitted over the network and processed at the control server. Fig. 8 presents a comparison of the average delay with respect to the increasing number of nodes in the DP. LS-CNM outperforms as compared to Kobo et al. [12] and OpenFlow because it provides the optimized placement of the controller to reduce the latency of flow rule generation. Moreover, the transmission time from the source to the destination is decreased because intermediate devices in the DP forward the data packets quickly based on the flow rules provided by the controllers frequently.

VI. Conclusion

The proposed work focuses on solving the controller placement and load imbalance problem in the distributed CP of the 6G IoE-based SDN-enabled WSN. LS-CNM is proposed to reduce the load of an overloaded controller using partial CN migration during ISAC among the 6G IoE devices. However, the latency is reduced using optimal placement of controllers inspired by CO whereas the initial controllers are identified using the graph theory-based AP method. The simulation result shows the effectiveness of LS-CNM by reducing the FCO by 84% and 71% in comparison with OpenFlow and Kobo et al. [12], respectively. Also, the partial CN migration maintains the load of controllers below the threshold value. The optimal placement of controllers improves the RTT of the proposed LS-CNM. Moreover, LS-CNM reports less delay in transmitting the data from source to destination as compared to the state-of-the-art approaches.

In the future, LS-CNM can be merged with AI in 6G IoE to predict and prevent potential issues like fault tolerance, and overloaded controllers in the network for reducing downtime of the CP. Additionally, there is room for further research in assessing the influence of dynamic network conditions and exploring the energy efficiency implications of the proposed method.

References

[1] X. Fang, W. Feng, Y. Chen, N. Ge, and Y. Zhang, "Joint communication and sensing toward 6G: Models and potential of using MIMO," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 4093–4116, Mar. 2023, doi: 10.1109/JIOT.2022.3227215.

[2] S. Verma, S. Kaur, M. A. Khan, and P. S. Sehdev, "Toward green communication in 6G-enabled massive Internet of Things," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5408–5415, Apr. 2021, doi: 10.1109/JIOT.2022.3227215.

[3] F. Liu et al., "Integrated sensing and communications: Toward dual-functional wireless networks for 6G and beyond," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 6, pp. 1728–1767, Jun. 2022.
[4] C. Ouyang, Y. Liu, and H. Yang, "Performance of downlink and uplink integrated sensing and communications (ISAC) systems," *IEEE Wireless Commun. Lett.*, vol. 11, no. 9, pp. 1850–1854, Sep. 2022.
[5] V. Tyagi and S. Singh, "GM-WOA: A hybrid energy efficient cluster routing technique for SDN-enabled WSNs," *J. Supercomput.*, vol. 79, pp. 14894–14922, Apr. 2023.
[6] V. Tyagi and S. Singh, "Network resource management mechanisms in SDN enabled WSNs: A comprehensive review," *Comput. Sci. Rev.*, vol. 49, Aug. 2023, Art. no. 100569.
[7] S. S. G. Shiny, S. S. Priya, and K. Murugan, "Control message quenching-based communication protocol for energy management in SDWSN," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 3, pp. 3188–3201, Sep. 2022.
[8] S. Moazzeni, M. R. Khayyambashi, N. Movahhedinia, and F. Callegati, "On reliability improvement of software-defined networks," *Comput. Netw.*, vol. 133, pp. 195–211, Mar. 2018.
[9] T. Abu-Ain, R. Ahmad, R. Wazirali, and W. Abu-Ain, "A new SDN-handover framework for QoS in heterogeneous wireless networks," *Arab. J. Sci. Eng.*, vol. 48, pp. 10857–10873, Mar. 2023.
[10] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN control: Survey, taxonomy, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 333–354, 1st Quart., 2018.
[11] A. Narwaria and A. P. Mazumdar, "Software-defined wireless sensor network: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 215, Jun. 2023, Art. no. 103636.
[12] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Fragmentation-based distributed control system for software-defined wireless sensor networks," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 901–910, Feb. 2019.
[13] A. Shirmarz and A. Ghaffari, "Taxonomy of controller placement problem (CPP) optimization in software defined network (SDN): A survey," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 12, pp. 10473–10498, 2021.
[14] S. Tahmasebi, N. Rasouli, A. H. Kashefi, E. Rezabeyk, and H. R. Faragardi, "SYNCOP: An evolutionary multi-objective placement of SDN controllers for optimizing cost and network performance in WSNs," *Comput. Netw.*, vol. 185, Feb. 2021, Art. no. 107727.
[15] G. Li, J. Wu, S. Li, W. Yang, and C. Li, "Multitentacle federated learning over software-defined Industrial Internet of Things against adaptive poisoning attacks," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1260–1269, Feb. 2023.
[16] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Efficient controller placement and reelection mechanism in distributed control system for software defined wireless sensor networks," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 6, 2019, Art. no. e3588.
[17] W. Wang, M. Dong, K. Ota, J. Wu, J. Li, and G. Li, "CDLB: A cross-domain load balancing mechanism for software defined networks in cloud data centre," *Int. J. Comput. Sci. Eng.*, vol. 18, no. 1, pp. 44–53, 2019.
[18] J. Wang, S. Zhang, W. Chen, D. Kong, X. Zuo, and Z. Yu, "Design and implementation of sdn-based underwater acoustic sensor networks with multi-controllers," *IEEE Access*, vol. 6, pp. 25698–25714, 2018.
[19] H. Babbar, S. Rani, A. K. Bashir, and R. Nawaz, "LBSMT: Load balancing switch migration algorithm for cooperative communication intelligent transportation systems," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 3, pp. 1386–1395, Sep. 2022.
[20] N. Aljeri and A. Boukerche, "An efficient heuristic switch migration scheme for software-defined vehicular networks," *J. Parallel Distrib. Comput.*, vol. 164, pp. 96–105, Jun. 2022.
[21] K. S. Sahoo et al., "ESMLB: Efficient switch migration-based load balancing for multicontroller SDN in IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5852–5860, Jul. 2020.
[22] R. Salam and A. Bhattacharya, "Efficient greedy heuristic approach for fault-tolerant distributed controller placement in scalable SDN architecture," *Cluster Comput.*, vol. 25, no. 6, pp. 4543–4572, 2022.
[23] C. Li, K. Jiang, and Y. Luo, "Dynamic placement of multiple controllers based on SDN and allocation of computational resources based on heuristic ant colony algorithm," *Knowl.-Based Syst.*, vol. 241, Apr. 2022, Art. no. 108330.
[24] L. Tian, A. Bashan, D. N. Shi, and Y. Y. Liu, "Articulation points in complex networks," *Nat. Commun.*, vol. 8, no. 1, pp. 1–9, 2017.
[25] H. L. Bodlaender, "On linear time minor tests with depth-first search," *J. Algorithms*, vol. 14, no. 1, pp. 1–23, 1993.
[26] R. Ramteke, S. Singh, and A. Malik, "Optimized routing technique for IoT enabled software-defined heterogeneous WSNs using genetic mutation based PSO," *Comput. Stand. Interfaces*, vol. 79, Jan. 2022, Art. no. 103548.
[27] Y. Cheng, J. Du, J. Liu, L. Jin, X. Li, and D. B. da Costa, "Nested tensor-based framework for ISAC assisted by reconfigurable intelligent surface," *IEEE Trans. Veh. Technol.*, vol. 73, no. 3, pp. 4412–4417, Mar. 2024.
[28] X. Li et al., "Physical-layer authentication for ambient backscatter aided NOMA symbiotic systems," *IEEE Trans. Commun.*, vol. 71, no. 4, pp. 2288–2303, Apr. 2023.
[29] S. S. Gill et al., "Modern computing: Vision and challenges," *Telematics Informat. Rep.*, vol. 13, Mar. 2024, Art. no. 100116.
[30] A. Montazerolghaem, "Software-defined Internet of Multimedia Things: Energy-efficient and load-balanced resource management," *IEEE Internet Things J.*, vol. 9, no. 3, pp. 2432–2442, Feb. 2022.
[31] A. Montazerolghaem and M. H. Yaghmaee, "Load-balanced and QoS-aware software-defined Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3323–3337, Apr. 2020.
[32] A. Montazerolghaem, "Software-defined load-balanced data center: Design, implementation and performance analysis," *Clust. Comput.*, vol. 24, no. 2, pp. 591–610, 2021.
[33] A. H. Alhilali and A. Montazerolghaem, "Artificial intelligence based load balancing in SDN: A comprehensive survey," *Internet Things*, vol. 22, Jul. 2023, Art. no. 100814.

Vikas Tyagi received the Ph.D. degree from the Department of Computer Science and Engineering from the Dr B R Ambedkar National Institute of Technology, Jalandhar, India, in 2024.

He is currently an Assistant Professor with the School of Computer Science Engineering and Technology, Bennett University, Greater Noida, India. His research interests include wireless sensor networks, software-defined wireless sensor networks, Internet of Things, cryptography, and information security.

Samayveer Singh (Member, IEEE) received the B.Tech. degree in information technology from Uttar Pradesh Technical University, Lucknow, India, in 2007, the M.Tech. degree in computer science and engineering from the National Institute of Technology, Jalandhar, India, in 2010, and the Ph.D. degree from the Department of Computer Engineering, Netaji Subhas Institute of Technology (University of Delhi), Dwarka, India, in 2016.

He is currently working as an Assistant Professor with the Computer Science and Engineering Department, National Institute of Technology. He has published more than 100 research articles in various international journals and conferences. His research interest includes wireless sensor networks, Internet of Things, data hiding, and information security.

Dr. Singh has been included in the list of Top 2% Scientists in the world ranking of 2021, which is released by Stanford University and Elsevier BV. He is serving as reviewer/member of editorial board for many journals/conferences.

Huaming Wu (Senior Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2009 and 2011, respectively, and the Ph.D. degree (Hons.) in computer science from Freie Universität Berlin, Berlin, Germany, in 2015.

He is currently a Professor with the Center for Applied Mathematics, Tianjin University, Tianjin, China. His research interests include wireless networks, mobile edge computing, Internet of Things, and complex networks.

Sukhpal Singh Gill received the Ph.D. degree in computer science from Thapar University, Patiala, India, in 2016.

He has been an Assistant Professor of Cloud Computing with the School of Electronic Engineering and Computer Science, Queen Mary University of London, U.K., since 2019. He has published in prominent international journals and conferences, such as IEEE/ACM TRANSACTIONS, IEEE INTERNET OF THINGS JOURNAL, IEEE/ACM UCC, and IEEE CCGRID. His research interests include cloud computing, edge computing, IoT, and energy efficiency.

Dr. Gill is serving as an Editor-in-Chief for *IGI Global International Journal of Applied Evolutionary Computation*, an Area Editor for *Cluster Computing Journal* (Springer), and an Associate Editor for IEEE INTERNET OF THINGS JOURNAL, *Internet of Things Journal* (Elsevier), *Transactions on Emerging Telecommunications Technologies* (Wiley), and *IET Networks*. For further information, see http://www.ssgill.me.