Metaheuristic-Based Approaches for Solving the Controller Placement Problem in Software-Defined Wireless Sensor Networks (SDWSNs)

Nivine Samarji

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Eastern Mediterranean University
June 2021
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Ali Hakan Ulusoy
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Assoc. Prof. Dr. Mohammed Salamah
Supervisor

Examining Committee

1. Prof. Dr. Cüneyt F. Bazlamaçcı
2. Prof. Dr. Mehmet Ufuk Çağlayan
3. Assoc. Prof. Dr. Adnan Acan
4. Assoc. Prof. Dr. Gürcü Öz
5. Assoc. Prof. Dr. Mohammed Salamah

ABSTRACT

Software-Defined Networking (SDN) is well-known as a master solver for many traditional networks restrictions. Network management flexibility is the SDN core characteristic that recently becomes a hot topic for many researchers to improve network performance. A critical issue arises specifically for the multi-controller SDN-based network, namely the controller placement problem (CPP), known as an NP-hard problem. Solving such problems in a reasonable amount of time is usually carried out using metaheuristic algorithms. This thesis presents different approaches for solving the CPP which are based on metaheuristic algorithms such as the genetic algorithm (GA) and the greedy randomized adaptive search procedure (GRASP). The first approach is a fault tolerance metaheuristic-based scheme (FTMBS) that we proposed for solving the CPP in wireless software-defined networks. The FTMBS is a multi-objective-based scheme aiming to maximize network connectivity and the load balance among controllers, minimize the network worst-case latency, and maximize the network lifetime in the presence of faulty nodes. In the presence of conflicting multi-objective metrics, the decision-maker or the network administrator decides on the compromise between these conflicting metrics. We defined the selection criteria for the number of SDN controllers ahead of time that achieves the targeted average percentage of network improvement. Simulations carried out for three and five controllers showed that three controllers were enough for such networks. We have shown the efficiency of the proposed FTMBS scheme under various percentages of faulty nodes as it has lower latency compared to the random distribution of controllers among cluster heads and the cluster-based network partition algorithm (CNPA). Also, we have verified the goodness of the solutions by

showing that the GA solutions have a good approximation to the Pareto optimal solutions provided by the Non-dominating Sorting Genetic Algorithm (NSGA-II). In addition, the proposed FTMBS scheme showed its superiority over various state-of-the-art schemes for different network performance metrics on the cost of having a bit higher complexity.

The second approach, namely the ALBATROSS scheme, is an energy-efficient strategy for WSNs, which is a modification of the FTMBS scheme. The albatross bird's dynamic soaring scheme is adopted in the cluster heads selection algorithm, and the selected cluster heads are taken as inputs for the heuristic algorithm to solve the CPP. Simulation results showed that the ALBATROSS scheme saves the network energy and outperforms other existing energy-aware schemes found in literature. However, it has a bit higher complexity than other schemes.

Besides, in Appendix A, we have provided an example of using the SDN technology in a practical environment other than WSNs, such as wireless body area networks (WBANs).

Keywords: Software-defined networking, controller placement problem, meta-heuristic algorithm, network performance enhancement.

ÖZ

Yazılım Tanımlı Ağ Oluşturma (SDN), geleneksel ağın karşılaştığı birçok geleneksel kısıtlamalara karşı ana bir çözücü olarak iyi bilinir. Ağ yönetimi esnekliği, birçok araştırmacı için ağ performansını geliştirme konusunda ilgi alanı haline gelmiş olan SDN temel özelliğidir. Çok denetleyicili SDN tabanlı ağ için kritik bir sorun olan ve NP-zor problem olarak da bilinen denetleyici yerleştirme sorunu (CPP) ortaya çıkmaktadır. Bu tür problemlerin makul bir sürede çözülmesi genellikle meta-sezgisel algoritmalar kullanılarak gerçekleştirilir. Bu tezde, genetik algoritma (GA) ve açgözlü randomize uyarlamalı arama prosedürü (GRASP) gibi meta-sezgisel algoritmalara dayanan CPP'nin çözümü için farklı yaklaşımlar sunmaktadır. İlk yaklaşım, FTMBS olarak adlandırılan kablosuz yazılım tanımlı ağlarda CPP'yi çözmek için önerdiğimiz hata toleransı meta-sezgisel tabanlı bir şemadır. FTMBS, ağ bağlantısını ve denetleyiciler arasındaki yük dengesini en üst düzeye çıkarmayı, denetleyicilerin kendileri arasındaki en kötü durum gecikmesini en aza indirmeyi ve hatalı düğümlerin varlığında ağ ömrünü en üst düzeye çıkarmayı amaçlayan çok amaçlı bir şemadır. Birbiri ile çatışan çok amaçlı ölçümlerin varlığında, karar verici veya ağ yöneticisi bu sözkonusu ölçümler arasındaki uzlaşmaya karar verir. Hedeflenen ortalama ağ iyileştirme yüzdesine ulaşan SDN denetleyicilerinin sayısı için seçim kriterleri önceden belirlenmiştir. Üç ve beş denetleyici için simülasyonlar yapılmış olup, bu tür ağlar için üç denetleyicinin yeterli olduğu belirlenmiştir. Önerilen FTMBS şemasının verimliliğini, çeşitli hatalı düğüm yüzdeleri altında, kontrolörlerin küme kafaları ve küme tabanlı ağ bölümleme algoritması (CNPA), ve rastgele dağılımına kıyasla daha düşük gecikme süresine sahip olduğunu gösterdik. Ayrıca, GA çözümlerinin, Baskın Olmayan Sıralama Genetik Algoritması (NSGA-

II) tarafından sağlanan Pareto optimal çözümlerine iyi bir yaklaşıma sahip olduğunu göstererek çözümlerin iyiliğini doğruladık. Ek olarak, önerilen FTMBS şeması, biraz daha yüksek karmaşıklığa sahip olma maliyetiyle farklı ağ performans ölçütleri için çeşitli son teknoloji şemalara göre üstünlüğünü göstermiştir.

İkinci yaklaşım, yani ALBATROSS şeması, FTMBS şemasının bir modifikasyonu olan WSN'ler için enerji verimli bir stratejidir. Ağ enerjisinden tasarruf etmeyi amaçlayan albatross kuşunun dinamik yükselme şeması, SDN denetleyicilerinin seçimleri için dikkate alınacak ağ küme kafalarını etkin bir şekilde seçmek için benimsenmiştir. Simülasyon sonuçları, ALBATROSS şemasının ağ enerjisini koruduğunu ve literatürde bulunan farklı mevcut enerjiye duyarlı şemalardan daha iyi performans gösterdiğini ortaya çıkarmıştır.

Ayrıca, Ek A'da, kablosuz vücut alanı ağları (WBANs) gibi WSN'ler dışında pratik bir ortamda SDN teknolojisinin kullanımına bir örnek sunduk.

Anahtar Kelimeler: Yazılım tanımlı ağ oluşturma, denetleyici yerleştirme problemi, meta-sezgisel algoritma, ağ performansı geliştirme.

DEDICATION

I would like to dedicate my work to my father's soul, mom, brothers, husband, and my world LARA.

ACKNOWLEDGMENT

I want to show my sincere appreciation to Assoc. Prof. Dr. MOHAMMED SALAMAH for his supervision, advice, and management from the beginning of my thesis work till this stage. He was my guiding mentor who did not hesitate even for a second to keep enriching me with his extraordinary experiences throughout the work. He played an essential role in being a psychological guide, ensuring that I do not lose this enthusiasm, especially during the publication stage. He was an easy-going proficient, a teacher, and a friend. His encouragement and support were endless and a true inspiration to think outside the box. I owe him a lot that even words are not enough to show him gratitude.

I also want to show appreciation to the department chair, Prof. Dr. AYBAY, and vice-chair, Assoc. Prof. Dr. BITIRIM, for their kind-heartedness and their generosity. Also, I want to show gratitude to the monitoring committee members for their helpful advice and assistance, especially Assoc. Prof. Dr. ADNAN ACAN for supporting and guiding me without any regrets. Also, not to forget to mention Assoc. Prof. Dr. GURCU ÖZ, for her involvements. I thank them all for their significant contribution and comments that added great quality to my thesis work. My thanks go to Prof. Dr. ADHAM MACKIEH, who did not hesitate to help and guide me. He is one of the best professors who endlessly gives support and advice.

Finally, I raise my chapeaux to every staff at the department mostly Mr. Erdal Altun, Mr. Mehmet Topal, Ms. Çiğdem Vudalı , Mrs. Deniz Gök, Mrs. Semiha Sakalli. They were like a second family whom I will never forget.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

API	Average Percentage of Improvements
BIP	Binary Interger Program
BS	Base Station
BSA	Balance State Awareness
BSS	Balance State System
BSW	Balance State Warranty
CCPP	Capacitated Controller Placement Problem
CH	Cluster Head
CNPA	Clustering-based Network Partition Algorithm
COLBAS	Cooperative Load Balancing Scheme
CPP	Controller Placement Problem
C_{root}	Root controller
DBCP	Density-Based Cluster Placement
DESS	Dynamic Energy Soaring Scheme
E2E	End-to-End
EA	Evolutionary Algorithms
EASM	Efficiency-Aware Switch Migration
EEFCA	Energy-Efficient Fault-Tolerant Clustering Algorithm
ERQTM	Energy-Efficient Routing and QoS Supported Traffic Management
FTMBS	Fault Tolerance Metaheuristic-Based Scheme
GA	Genetic Algorithm
GCEEC	Gateway Clustering Energy-Efficient Centroid
GRASP	Greedy Randomized Adaptive Search Procedure

LFC	Load distribution of Faulty Controller
ms	Milliseconds
NOS	Network Operating System
ns	Nanoseconds
NSGA	Nondominated Sorting Genetic Algorithm
ONOS	Open Network Operating System
O	Time and Space Complexity
pps	Packets Per Second
PSA	Pareto Simulated Annealing
QoS	Quality of Service
ROC	Rank Order Centroid
SA	Simulated Annealing
SDN	Software-Defined Networking
SDWBAN	Software-Defined Wireless Body Area Network
SDWSN	Software-Defined Wireless Sensor Network
SOF	Sensor OpenFlow
SSOA	Salp Swarm Optimization Algorithm
TDMA	Time Division Multiple Access
WBAN	Wireless Body Area Network
WSN	Wireless Sensor Network
µs	Microseconds

Chapter 1

INTRODUCTION

1.1 Software-Defined Networking

Nowadays, the world represents a vast linked digitalized society with the help of the internet, making almost everything accessible and connected. The traditional internet fundamental infrastructure embodies the vertical integration between the control and data planes residing on the same device. Hence, controlling and managing the network becomes a challenging task with the bulk network growth. Moreover, this vertical integration sets further restrictions on network configuration for predefined policies and responding to various network changes. Initially, Software-defined networking (SDN) idea is proposed to handle several challenges in wired IP networks, such as the network complexity and the difficulty of configuring the network behavior [1]. Due to the challenges mentioned above, thumbs are pointed toward the SDN technology. SDN is a new emerging paradigm that promises to change this state of affairs [2] by simply releasing the vertical integration, separating the network's control logic (control plane) from the network devices (data plane), and allowing flexibility in managing policies [3] and reconfiguration of the network. As a result of this decoupling, the network devices become simple forwarding devices, and the extracted control logic is inserted at a centralized controller or network operating system (NOS) [4]. The routing decisions and policies are handled by the controller and deployed in the switch's flow table. Figure 1 shows the traditional network structure versus SDN network structure.

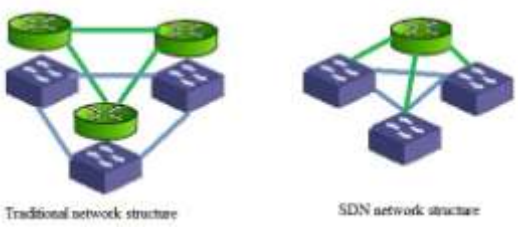

Figure 1: Traditional Network Structure vs. SDN-Based Network Structure [2]

1.2 The Use of SDN in WSN

WSN is made up of several cheap and low-powered stationary or mobile sensor nodes. Currently, WSN is integrated with industrial areas and with many innovative technologies such as smart building (Heating, Ventilation, and Air conditioning (HVAC)) systems to comply with the user requirements. These technologies aim at achieving energy-efficient resource utilization. The core purpose of WSN is to sense and collect event-driven data in a particular field. However, the main WSN limitations [5], [6] are the changes in the network topology and the limited energy supply of the sensor nodes [7], [8], which in case of quick depletion can cause network disconnection. Hence, well-managed strategies and protocols can save energy as a scarce resource. In addition to energy, the integration of control and data plane makes the network extension and management complicated and costly [4], [1]. The first implementation of SDN in WSN is presented in [9], where the authors proposed the Sensor OpenFlow (SOF) as the communication protocol between the data plane and control plane. The data plane corresponds to the sensor nodes that are flow-based packet forwarding elements, receiving the forwarding rules from the control plane that corresponds to the controller, known as the network brain which handles all the decisions. SDN is characterized by releasing management challenges in wireless sensor networks.The integration of the SDN paradigm in the WSNs (SD-WSN) [10], [5], [11], [12], [13] has recently established widespread concentration

where researchers focused on applying SDN in WSNs [1] in regards to architecture and network topology [9], [14], routing protocols [15], [16], [17], [18], [19], node scheduling and energy-saving [20], [21], data transmission and load balancing [22], [23] as well as network security [24]. The network topology in SDWSN refers to the physical layout of various sensor devices interconnected by transmission media [1]. The SDWSN network topology can be both fixed and mobile [25]. Fixed topology is either centralized topology deployed in a small-scale network or distributed topology deployed in a large-scale network. This thesis focused on distributed SDWSN, where the sensor nodes are arranged into clusters according to a distributed SDWSN model. The SDN main controller is deployed at the gateway, i.e., the sink that manages and coordinates the sub-controllers. The cluster heads manage sensor nodes in each cluster where each cluster is considered as the zone or domain of the SDN sub-controllers, as shown in Figure 2.

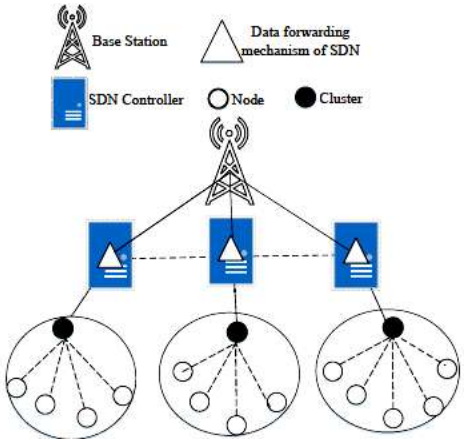

Figure 2: Distributed SDWSN

Although SDN can release all the challenges faced with traditional networks; a severe factor that an SDN-based network needs to provide is the reliability [26], [27]. In other words, an SDN-based network should not be affected by a single point of

failure. Consequently, this will lead to total network breakage since the control plane provides all the network information to the data plane; hence, any disconnection between them ends the network performance. There are different studies of using the concept of SDN in WSNs as shown in Figure 3. In this thesis we have followed a multiobjective approach in which we considered energy efficiency (lifetime, and clustering), and reliability issues to solve the CPP.

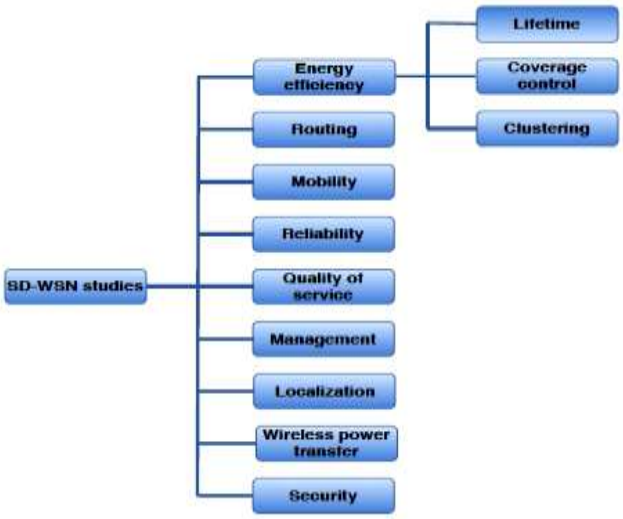

Figure 3: SDN-Based Approaches for WSNs [27]

Based on our knowledge, few frameworks are used for implementing SDN-based WSN such as SDN-WISE [28], SDWN-ONOS [29], and TinySDN [30], [31]. However, these frameworks have several drawbacks, i.e., their codes do not comply with the WSN firmware, making it inapplicable to the existing WSNs.

SDWN-ONOS is the first to provide the Open Network Operating System (ONOS) [5], which uses the existing SDN controller in the wired network. To make ONOS relevant in SDN-based WSNs, WSN devices such as SDN-WISE Emulated mote and SDN-WISE Emulated sink [28] were adopted from SDN-WISE. However, these devices were designed based on a different standard than IEEE 802.15.4 standard,

making their integration in WSNs inappropriate. Hence, further investigations and studies should be carried out to make SDN-WISE, SDWN-ONOS, and TinySDN appropriate to be implemented in WSNs. In fact, integrating the concept of SDN with WSNs has been investigated by many reasearchers in literature. Table 1 illustrates some of these state-of-the-art frameworks.

Table 1: SDN Frameworks in WSNs [32]

Year	SD-WSN	Features	Simulation	Performance Evaluation	WSN Compa-rison
2012	Sensor OpenFlow [9]	Propose a concept of SD-WSN	No	No	No
2012	SDWN [5]	Propose a concept of SD-WSN	No	No	No
2014	Smart WSN-SDN [14]	Propose a concept of SD-WSN	No	No	No
2014	TinySDN [30]	Provide an SD-WSN framework	Cooja	Response time, memory	No
2015	SDN ECCKN [33]	A centralized sleep scheduling algorithm	MATLAB	Network Lifetime	Yes
2015	SDWN ONOS [29]	Provide an SDN-IoT framework	Cooja, Mininet	No	No
2015	Multi-Task SDSN [34]	A centralized algorithm to optimize energy efficiency	Gurobi Optimizer	Sensing rate, reschedulin g time, power efficiency	No
2015	SDN-WISE [28]	Provide an SD-WSN framework	Cooja	RTT, efficiency, response time	No

2016	Routing SDWSN [35]	Energy-efficient algorithm for SD-WSN	MATLAB	Network Lifetime	Yes
2016	WARM [36]	Provide an SD-WSN framework	Cooja	Comm. overhead, memory	No
2016	SDN-TAP [37]	Provide an SD-WSN framework	Cooja	Delay, packet loss	No
2016	SDWSN-IoT [38]	Propose a concept of SDN-IoT	No	No	No

1.3 Controller Placement Problem

The controller placement problem (CPP) incorporates the number of controllers [39] and includes the controllers' proper positions in an SDN-based network. Researchers have widely focused on solving the CPP to enhance the overall network performance and meet acceptable QoS levels. Efficiently allocating switches to controllers without causing overwhelming the controllers was presented in [40], aiming to minimize network latency, maximize the network fault tolerance and reliability, and minimize the number of controllers and node consumption energies. Therefore, the number and location of SDN controllers known as CPP can dramatically affect the overall network performance. Heller et al. in [41] investigate a significant issue dealing with the number of SDN controllers to be implemented at particular topology positions to meet the network requirements. For instance, a good placement aims to reduce the propagation latency among the nodes and SDN controllers in wide area networks (WAN). It is good to note that in WAN, the edges' weights represent the propagation latencies which can be obtained using the Haversine formula [42], [43], [44]. Hence,

in the case of having two disjoint islands, the best controller placement which reduces the average latency is at the midpoint of the distance between the two islands. However, in the case of two controllers, the placement of one controller at each island will remarkably reduce the total network latency.

As stated previously, the main three layers of SDN are the data plane, control plane, and application plane. The control plane is the brain of the network [45]. All decisions, policies, and forwarding rules are defined by the controller, so that the data plane comprises only policy recipients forwarding devices. The application logic [2] is set by the application plane located on top of the control plane. Upon existence, the SDN-based network was made up of only one controller. Researchers then found out that one SDN controller can't cope with the fast network growth, and as a result, can't handle all the control requests coming from the data plane. This issue is known as scalability. Besides scalability issues, reliability problems caused by a single point of failure were behind focusing on the use of multi-controllers [46] in SDN-based networks. Remarkable network performance improvement was recorded with the implementation of a multi-controller. Nevertheless, a critical factor accompanied by the multi-controller implementation which directly affects the overall network performance is the controller placement problem (CPP) [41].

Solving the CPP is not a straightforward approach, since it defines the necessary number of SDN controllers and their best positions in the network. The number of SDN controllers is essential since excessive use of SDN controllers affects the cost of implemention and leads to unnecessary delays [47] caused by many control requests. Since CPP is an NP-hard problem [46], solving such problems is usually carried out using evolutionary algorithms. Therefore, solving the CPP in a multi-

controller-based network is actually finding both the controllers' optimum number and location.

The issue of CPP was investigated firstly by [41]. The authors focused on latency issue as a critical factor affecting real-world network performance. However, the load balancing factor between the controllers was missed in their work. In wireless networks, the communication link and available bandwidth are shared by the data and control planes, leading to unnecessary delays in the presence of load imbalance. Hence, the presence of a load balancing algorithm highly limits the occurrence of unnecessary delays. Accordingly, the network performance is directly affected by network reliability and network delay exposed by the propagation and queuing latencies [48].

Authors of [41] have motivated researchers to provide network solutions for improving network performance. Nowadays, SDN is integrated with Google, Facebook, cloud computing, and many applications [45], [11].

Evolutionary algorithms are efficient optimization methods that give near-optimal solutions satisfying the specified network constraints. Hence, evolutionary algorithms are widely used to solve the CPP in an SDN-based network where CPP is considered an NP-hard problem [49], [50], [51], [52]. When applying evolutionary algorithms, researchers usually define the network constraints and the objective function satisfying the defined constraints.

Cisco APIC [53] uses a minimum of three SDN controllers. The analysis showed that using k controllers instead of one controller will not reduce the latency by 1/k factor

[41]. In networks where topology changes frequently, it is found that the controller placement is very significant to minimize the packet propagation latency [54]. Ideally, the number of controllers should be minimized for cost issues, while minimizing communication latency at the same time. Network latency includes the time it takes to transmit the packets, time spent in the switch buffer before processing (queueing time), and the time it takes for the controller to process the packets. Latencies do not weigh equally for different networks. For example, in WAN, only the time it takes for the packets to be transmitted and the time it takes for the controller to process the packets are considered [5]. Packet transmission latency is directly affected by the distance between controllers and switches. Therefore, in assigning switches to controllers, the shortest path is usually considered. The controller's processing latency is affected by loads of the controllers [48]; hence, this justifies why researchers are being attracted by a controller placement solution that leads to flow balance among controllers.

1.4 The Aim of the Study

In Literature, necessary conditions are absent and should be considered while solving the CPP in an SDN-based network. The presence of node and path failures and the steady-state network attainment after executing the load balancing algorithm should be considered. These two conditions are taken into consideration in this study. We used metaheuristic algorithms to solve the CPP, aiming to maximize the network connectivity, balance the load, minimize the network delay, and increase the network lifetime. Specifically, we applied the genetic algorithm and GRASP algorithm [55], [56] to solve the CPP. Keeping the network in a steady-state is achieved by proposing the Balance State System (BSS) algorithm which balances the loads among the controllers and ensures no overwhelming controller(s) exists after

distributing the load of the faulty controller(s). Besides the above-mentioned aspires, an important aim to be achieved in SDN-based WSN is saving network energy and enhancing network lifetime. This aim is achieved by using the ALBATROSS algorithm described in Chapter 6. Shifting towards the software-defined wireless body area network, an important metric to be achieved is network reliability, ensuring a delay-free transmission of the emergency data. Hence, as a practical example, we proposed the ERQTM algorithm, presented in Appendix A.

1.5 Thesis Contributions

The contributions of our work-study are:

a) Achieving network reliability by applying the spatial clustering algorithm that produces alternative paths in the presence of path failure or node failure.

b) Applying the Genetic algorithm and GRASP algorithm to provide near-optimal solutions in a reasonable amount of time, satisfying various network constraints in the presence of faulty nodes, and optimizing the network Quality of Service (QoS).

c) Considering a variable flow rate in the proposed BSS algorithm that achieves a dynamic load balance among controllers by adopting the network traffic changes. Thus, improving the network efficiency. Besides network efficiency, BSS achieves the network steady-state with minimum latency.

d) Carrying a comprehensive study for solving the CPP that incorporates the presence of faulty nodes and a tradeoff between different conflecting network objectives.

e) Introducing the ALBATROSS algorithm which had a positive impact on the WSN's lifetime enhancement.

f) Proposing the ERQTM algorithm which achieves the SDWBAN reliability.

Chapter 2

LITERATURE REVIEW

The related literature is reviewed in five sections. In section 2.1, the studies on solving the CPP using evolutionary algorithms have been mentioned. Solving the CPP while focusing on load balancing among different controllers is showed in section 2.2. Solving the CPP while focusing on decreasing the network latency is given in section 2.3. Solving the CPP while focusing on increasing network reliability and energy efficiency is presented in section 2.4. Finally, section 2.5 mentions the studies that focus on enhancing the network lifetime while solving the CPP.

2.1 Integrating Evolutionary Algorithms in Solving the CPP

Different literature metrics are alleged to have a direct influence on the placement of the SDN controllers. These metrics are included in multi-objective approaches to solve the CPP [1], [4], [57], [58]. The main metrics include network latency [59], [60], network management [27], reliability and resilience [61], deployment cost [62], and energy consumption [63], [64]. In the network management, the control messages between controllers and switches are transmitted in a dedicated channel (out-band mode) and are generally small-sized flows in comparison to the dense flows in the data plane. The placement of controllers significantly affects the metrics [58], [65] mentioned above. The CPP is viewed as a multi-objective combinatorial optimization problem (MOCO) [66]. Solving this multi-objective problem is usually achieved by finding the Pareto-frontier [67], where a decision-maker selects the most

appropriate solution which is based on the type of studied environment. This multi-objective combinatorial optimization problem can efficiently find the Pareto-optimal in a reasonable amount of time in small to medium-size networks. However, the drawback lies when the network size to be analyzed is large, since the time is directly proportional to the network size, i.e., placing k controllers in n nodes is a combinatorial problem $\binom{n}{k}$ placement [45]. For this reason, heuristic approaches are used to solve multi-objective problems [45]. Mohanty et al. [68] considered the propagation latency and the cost metrics using a modified version of the genetic algorithm to solve the CPP. However, they have used a fixed cost of placing a controller at a specific location, which lacks further explanation. Sanner et al. [69] have also solved the CPP based on evolutionary algorithms to maximize the average cluster connectivity and balance the load among clusters. They have conducted their algorithm based on NSGA-II [70] framework that deals with CPP. Their proposed GA lacks a fundamental operator, the crossover operator, where they have claimed to consider for future work. Jalili et al. [52] solved the CPP based on NSGA-II [71], focusing on optimizing the inter-controller and intra-controller latencies. Champagne et al. [49] have proposed a multi-objective genetic algorithm for the CPP that aims to minimize inter-controller latency, load distribution, and the number of controllers with fitness sharing. The proposed GA approach provides diversity in fitness value for different solution spaces, and the diversity provides more solution spaces with various fitness values. One reason is that the local search improves the population quality by producing different solution in every iteration [72]. Therefore diversity in their algorithm is critical since it can provide various choices, especially for dynamic network reconfiguration. Their algorithm also provides adaptive solutions to real network architectures such as the United States backbone and Japanese backbone

networks. Lokesh et al. in [73] have proposed node fault detection and a recovery scheme for the CPP based on a genetic algorithm. Their scheme aims to eliminate faulty nodes by forcing the sleep mode when assigning the controllers based on energy and link efficiency. Another heuristic approach based on Simulated Annealing (SA) was proposed by Lange et al. [51] to solve the CPP. Their approach uses Pareto Simulated Annealing (PSA) and aims to minimize the average latency between controllers. The PSA algorithm explores 2.5% of the search space. Authors in [74] presented latency and cost-aware controller placement dynamic optimization algorithms, namely Salp Swarm Optimization Algorithm (SSOA). Their algorithm dynamically checks the optimum number of controllers by evaluating the network load changes, which puts controllers to sleep mode to decrease the communication overhead in case of a low-load network or adds controllers to handle the network's load in the presence of an overloaded controller. They conducted their search study on large-scale SDN networks. Hock et al. [50] have proposed a resilient-based Pareto-optimal controller placement method considering different factors such as latency and failure resilience. Their results showed no optimal value for both latency and failure resilience when considered simultaneously; instead, a tradeoff should be considered. Kwon and Kang [75] have proposed a genetic algorithm-based metaheuristic scheme to balance the controllers' load. Their load balancing scheme has a triggering factor which is a specific load threshold. Whenever the system detects load imbalance revealed by the load threshold, the load balance scheme is executed.

2.2 Solving the CPP Based on Load Balancing

Liao et al. [76] solved the CPP by presenting a density-based cluster placement (DBCP) algorithm which takes each controller's density into account. DBCP

separates the network into several connected sub-networks and assigns a controller for each cluster to achieve high network connectivity. In other words, DBCP converts the multi-controller placement problem into a single controller placement problem. DBCP can be easily implemented as it is mainly based on one parameter, the distance threshold. An increase in the number of clusters is related to either too small or too large threshold. Therefore, the distance threshold should be set as [0.3-0.5] times network diameter, and in their case, the distance threshold is set to 0.3 times network diameter.

Selvi et al. [77] presented a Cooperative Load Balancing Scheme for Hierarchical SDN Controllers (COLBAS). In their scheme, one controller is assigned as a super controller to manage other controllers' flow requests. When the super controller detects flow imbalance, i.e., the flow requests exceed an upper bound threshold, it uses a greedy algorithm to reassign different flow setups to proper controllers and redistribute the flows to reach a lower bound threshold. Installation of allocation rules on switches for load balancing is achieved by keeping a low-cost reassignment, i.e., assigning the flow to the lowest cost controller till the lower bound is met.

Cui et al. [48] have investigated multiple overloaded controllers and developed a load balancing strategy based on the controller's response time, where real-time of the controller's response variation is considered. By selecting the appropriate response time threshold, overloaded controllers could be identified if their response time is above the threshold, and non-overloaded controllers could be identified if their response time is below or equal to the selected threshold. By identifying the overloaded and non-overloaded controllers, the authors applied the migration algorithm. The heaviest switch belonging to the most prominent response time

controller is migrated to the lowest loaded controller iteratively. Their migration algorithm deals with multiple overloaded controllers simultaneously to palance the load. However, their strategy can't be considered as a bottom line for checking overloaded controllers, for instance, a controller could be faulty and not overloaded. In this case, the scheme can't work well. Also, their scheme lacks further investigation for the migration cost, which they left for future work.

In the case of a controller failure, achieving high availability is a challenging task. Rao et al. [78] presented a multi-controller cluster-based scheme where multiple controllers are organized in clusters form. In each cluster, one primary controller handles the work, and others synchronize the controller state information, including network topology, network services, applications and data synchronization. If a primary controller fails, the election strategy will be executed on the other controllers, and the one with the highest priority will be elected as the primary controller. Besides the election, the load balancing strategy is executed on the OpenVswitch which distributes the load on different servers in a round-robin fashion to prevent overwhelmed servers.

Yao et al. [78] considered both the SDN controllers' capacity and the maximum latency, so their algorithm is an alternative to the capacitated k-center problem (Capacitated Controller Placement Problem-CCPP). Their algorithm is based on the Integer Programming model to find the minimum number of needed controllers. However, their approach has a bit high time complexity. Their results show that a reduction in the number of controllers and the most capacitated controller load can be achieved for each placement.

Yao et al. [65] have introduced a new HybridFlow approach to use multi-controller for large wide-area wireless networks. They have proposed a balancing scheme based on double thresholds where controllers are arranged in many clusters. A super controller controls all these clusters. Each controller checks the load balance in its cluster and communicates with its neighboring controller to balance the load. Whenever the cluster load exceeds a specific limit, controllers in this overloaded cluster will transfer all requests to the super controller, which transfers the excessive load to other non-overloaded clusters. HybridFlow reduces the super controller overhead requests by letting the controllers manage the load within the cluster. Only the super controller will manage load balance in case one of these clusters is overloaded. Their simulation showed that the HybridFlow outperforms BalanceFlow [79] by reducing the super controller overhead requests and workload. Generally, the load among controllers is not static; it may vary according to spatial and temporal variation in traffic conditions. For that reason, Dixit et al. [80] presented a load balancing mechanism which adapts to traffic variations. Their algorithm keeps monitoring the load conditions among controllers. If load imbalance is detected, switches automatically migrate from the overloaded controller to the less loaded controller. The algorithm extends or shrinks the controller pool whenever necessary. That is, if the aggregate traffic load is higher/smaller than controller capacity, the controller pool scales down/up by adding a new node to the pool or removing a node from the pool. Their simulation results are pretty efficient regarding the response time, which lasts for 2ms even when the load rate increases or decreases.

In the sense of a multi-objective scheme, Ruiz-Rivera et al. [81] have proposed an efficient approach to balance the load among SDN controllers. Their approach limits

the delay to an upper bound and saves the energy by turning off as many links as possible, keeping the connection available among switches and controllers.

Yang and Wang in [82] proposed a path selection k-Dijkstra based algorithm for the load balancing module which can adapt to traffic changes on one side and reduce network delay and packet loss rate on the other side.

Killi and Rao in [83] solved the CPP by considering the load of controllers, latency, and network reliability revealed by the controller failure avoidance ahead of time. However, their strategy's drawback is the communication overhead done by switches that repeatedly send packet-in messages to the nearest controller without knowing its status. Another load balancing mechanism was introduced by Yu et al. [84], where each switch is connected to one master controller and several slave controllers. Whenever the master controller becomes overloaded, the switch with the highest flow rate is chosen to migrate to one of the slave controllers based on the recipient controller's highest cost, keeping the targeted controller's load below a specific threshold. Their results showed an increase in the throughput within a reasonable amount of time to complete the load balancing mechanism.

Hu et al. [85] solved the CPP by considering both the delay and load balancing concurrently. They claimed that an insignificant increase in the load is caused while optimizing the delay. They have suggested the use of a heuristic algorithm to optimize the delay along with the load balance for future work. Another efficient load balancing algorithm presented by Hu et al. [86] focused on the migration cost that the switch encounters when migrating from its associated overwhelmed controller to a less loaded one. They claim that the switch migration may increase the

recipient controller's response delay and decrease the throughput. By taking into consideration, the migration cost that represents the added load on the recipient controller, their proposed efficiency-aware switch migration (EASM) strategy results in load-balanced controllers, in addition to high controller throughput and low migration cost.

2.3 Solving the CPP Based on Decreasing Latency

Hu et al. in [87] have addressed the controller placement problem based on the network energy consumption performance metric. The control paths and capacity of controllers are modeled as binary integer program (BIP). In their model, the network energy consumption is minimized under the control paths' delay and controllers' loads' constraints. Due to the high complexity of BIP, the authors have suggested the use of a heuristic algorithm, specifically the genetic algorithm, to find the near-optimal solution. Results conducted on different topologies with various controllers' numbers reflect the indirect proportion between the number of controllers and the number of control path links. Also, results have shown that delays on all control paths satisfy the delay threshold, in addition to energy savings. However, their model aimed at energy saving in the control plane, and discarded the forwarding plane, which they claim to include as a future work for an energy-aware SDN model.

Abdelaziz et al. in [88] presented cluster-based distributed SDN controllers where three controllers are placed in a cluster; one of them is selected as the primary controller, and the others are backup controllers to assure the reliability and availability of the network. Their experiments show that when the number of switches exceeds 75, their algorithm reduces the latency from 8.1% (when no

clustering is done on the distributed controllers) to 1.6% when clustering exists. Also, the packet drop is reduced from 3.99% to 3.53%.

Another latency-based improving algorithm is presented by Wang et al. [89], where different aspects of latency have been considered that can affect the network performance. For instance, they have suggested using the Clustering-based Network Partition Algorithm (CNPA). The CNPA partitions the network to reduce the overall end-to-end latency, unlike traditional clustering algorithms. In such algorithms as k-mean and k- center, the end-to-end latency can't be reduced with the increase in the number of subnetworks. In addition to the end-to-end latency, they have investigated the queuing latency for switches modeled as M/M/m queueing system. Accordingly, they placed controllers at switches and iteratively increased the number of controllers so that the maximum latency is upper bounded by a given threshold. They found out that when eight controllers are deployed, the average total latency achieved by their algorithm is almost three times smaller than those achieved by k-means and k-center algorithms. The authors used the Haversine formula to find the shortest path between nodes instead of the Euclidean distance where the physical links may not exist in the path of the Euclidean distance between two nodes [90], and the shortest path distance between nodes is based on Dijkstra's theorem [91].

2.4 Solving the CPP Based on Reliability and Energy-Efficiency

Many researchers have handled the controller placement problem focusing on increasing reliability and allowing the overall system to be resilient to connectivity failure. Heuristic approaches such as l-w greedy [92] and simulated annealing have been adopted in [27], [92]. Authors in [27] have represented the reliability metric as the percentage of control paths' failure and as an expected percentage of valid control

paths where switches' failures vary within [0.015-0.025], and links failures vary within [0.035-0.045]. The authors in [92] have represented the reliability metric same as in [27] with different intervals as [0-0.02] and [0-0.04] respectively. Although, the reliability increases with the number of controllers, however, deploying an excessive number of controllers will have a negative impact on the reliability [92]. Therfore, authors in [92] illustrated that the number of controllers varies in the interval [0.035*n*-0.117*n*] where *n* is the number of nodes in the network. They have used the simulated annealing method to solve the controller placement problem. They have compared their results with the Brute Force search [93] technique which is regarded as the most straightforward meta-heuristic technique that works well and gives optimal results in limited size problems. Their findings show a decrease in reliability metric when optimizing the latency and an increase in latency when optimizing the reliability metric. These were their findings when three controllers are used. However, in the case of one controller, optimizing the latency leads to optimized reliability.

Since the network failure leads to a disconnection between control and data planes, achieving reliability for the controller placement problem was the aim of Zhang et al. [61]. The authors presented an algorithm to reduce the connectivity failure between controllers and switches based on a minimum cut (min-cut) algorithm to maximize network resilience to failure. Their proposed method showed better reliability improvements than greedy and random schemes. Hu et al. [94] focused on real topologies and defined reliability as the expected percentage of control path loss which is proven to be NP-hard. They have shown a tradeoff between network latency and network reliability.

2.5 Solving the CPP Based on Network Lifetime Enhancement

Hu et al. [87] focused on optimizing the network energy saving in solving the CPP. By simulations, near-optimal solutions are achieved. The authors adopted the GA [94] to optimize the average connectivity and the network balance to solve the CPP.

Qureshi et al. [95] proposed a load management scheme named as Gateway Clustering Energy-Efficient Centroid (GCEEC) to address the load burden issues caused by sensor nodes which relay their transmission data to those that are close to the base station. In their scheme, the CHs are selected from the centroid position, and the gateway nodes are selected from CHs, aiming at transmitting the data of the overwhelmed CHs to the base station. The experimental results showed that the proposed GCEEC scheme is an energy-efficient algorithm that showed better performance than other state-of-the-art schemes.

Nitesh et al. [8] proposed a fault tolerance and energy utilization-based scheme for large-scale networks. The proposed scheme named as Energy-Efficient Fault-Tolerant Clustering Algorithm for Wireless Sensor Networks (EEFCA) is formulated regarding the distance between the sensor nodes and the base station, residual energy, and the number of sensor nodes in each cluster. In their scheme, each sensor node calculates the cost of joining a relay node close to the base station whenever its associated CH is faulty. Accordingly, it sends the calculated info to the relay node, which transmits it to the base station. The experimental results showed that the EEFCA scheme is an energy-efficient algorithm that outperforms other schemes in the literature. However, the proposed scheme requires various calculations done by

the relay nodes for choosing the best CH for communication, which consumes a lot of energy.

Chapter 3

THE PROPOSED FAULT TOLERANCE METAHEURISTIC-BASED SCHEME (FTMBS)

This chapter presents a detailed description of the proposed system approach. In section 3.1, a description of the problem is presented in detail. Section 3.2 describes the methodology used to solve the CPP, which incorporates a k-way spectral clustering technique described in section 3.2.1. Section 3.3 presents the network structure with the related assumptions. Finally, section 3.4 presents the system description and the various network performance metrics to be optimized in solving the CPP.

3.1 Problem Description

Software-defined networking is a new technology that was initially implemented in wired networks and data centers. Afterward, and due to the flexibility it can offer for network management, this new technology motivated many researchers to implement it in wireless networks. TinySDN [30], SDWSN [5], and much more in Literature [1], [4] are examples of such implementation. Some network performance metrics are affected by sharing the communication link and network bandwidth among the data and control planes, such as network delay and network throughput. On a given link, both control and data requests are sent, causing unnecessary network delays. One promising solution for restricting the delay is providing an efficient load balance algorithm in a multi-SDN controller-based network [96].

As stated before, a critical issue arises when multi-controllers are present in an SDN-based network, the CPP, which is an NP-hard problem [2]. Solving such problems is usually done by applying evolutionary algorithms [51], [52], [69], [73], especially for large-scale networks. Evolutionary algorithms are optimization algorithms that provide near-optimal solutions in a reasonable amount of time. Identifying the problem along with the objectives and the network constraints are typically done ahead of time. Since the number of controllers affects the network performance, authors in [27], [2], [92], [94] claim that deploying few numbers of controllers or excessive number of controllers will reduce the reliability of the system.

The most important metrics affecting the overall network performance are reliability and latency. Latency includes transmission, propagation, and queuing delay [89]. Most researchers in literature did not consider the presence of faulty nodes. Hence, in this thesis, we have investigated the network performance when using SDN in WSNs in the presence of faulty nodes. We have solved the CPP by proposing a Fault Tolerance Metaheuristic-Based Scheme (FTMBS) [97] for forecasting-based monitoring environments [98]. The proposed scheme incorporates the use of both the Genetic algorithm [49] (GA) and the Greedy Randomized Adaptive Search Procedure (GRASP) algorithm [55], [99]. We have ensured to achieve the network steady-state by proposing the Balance State System (BSS) in the presence of faulty nodes. Our proposed scheme gives the network operator the freedom to decide on the tradeoff among different competing multi-objectives. The multi-objective-based algorithm optimizes the network connectivity, balances the load, minimizes the network delay, and optimizes the network lifetime.

3.2 Methodology

The proposed FTMBS scheme optimally solves the CPP and maximizes network fault resilience by considering the link efficiency and the node's remaining energy. Thus, the FTMBS scheme can be seen as an energy-efficient and fault-avoidance scheme. We investigated the tradeoff between the cost and the number of controllers even though the network performance shows slight improvement when the number of controllers increases. We followed what Cisco has declared about using odd numbers [53] of SDN controllers. We initially considered three SDN controllers to avoid the single point of failure, and then we considered five controllers. We carried out various network performance comparisons when using three and five controllers and decided on using three controllers in a network consisting of 500 sensor nodes in a field of 200x200 m^2.

3.2.1 K-way Spectral Clustering

One of the critical challenges in wireless sensor networks is the sensor nodes are empowered with a very limited battery, and recharging the battery is almost impossible. As known, energy saving has a direct positive impact on the overall network lifetime. To address this issue, Jorio et al. [100] have proposed a new clustering algorithm in WSN based on spectral clustering [101], [102], and residual energy. Their algorithm determines the optimal number of clusters and determines the cluster heads, which is somehow vice the conventional clustering methods such as LEACH. The K-Way spectral clustering [103] determines the number of clusters by considering the network field area and the energy consumption caused by the free space model amplifier and the multiple attenuation model amplifier. Based on the Laplacian matrix, eigenvalues and eigenvectors are determined. Then clustering is done using K-mean on the matrix consisting of K-eigenvectors of the Laplacian

matrix's largest K-eigenvalues. The last step is selecting the cluster heads, which differs from traditional random selection process. Instead, two critical parameters are considered when selecting a cluster head: the sensor nodes' residual energy and node id. Therefore, the cluster head is at a random position in the cluster in each round. As a result, their approach caused a significant reduction in the total consumed energy of each round and a noticeable elongation in the network lifetime. In this thesis, the motivation to apply the k-way spectral clustering is two-fold. First, applying this type of clustering proves to cause a noticeable reduction in the network consumed energy which is the most desired objective in WSNs. Second, spectral clustering provides several disjoint clusters, and alternative paths exist for each node in the network. Therefore, in the presence of faulty nodes or paths, the transmission of messages to the destination can still be performed via alternative paths. Since we have considered various percentages of faulty nodes and paths, alternative paths will boost the network reliability.

3.3 Network Structure

In this thesis, we focused on a multi-domain network, where 500 sensors are deployed randomly in a field of 200m × 200m for forecasting-based monitoring environment as in [98] where three controllers are present at the beginning; one placed at the sink and two others to be selected from the network cluster heads. Each domain is controlled by a specific controller, where the network state is shared between them. The controllers are assumed to be connected via Ethernet cable, as shown in Figure 4. The root controller at the sink, considered to be failure-free, executes the necessary algorithms for achieving the network steady-state and ensuring the load balance among the controllers.

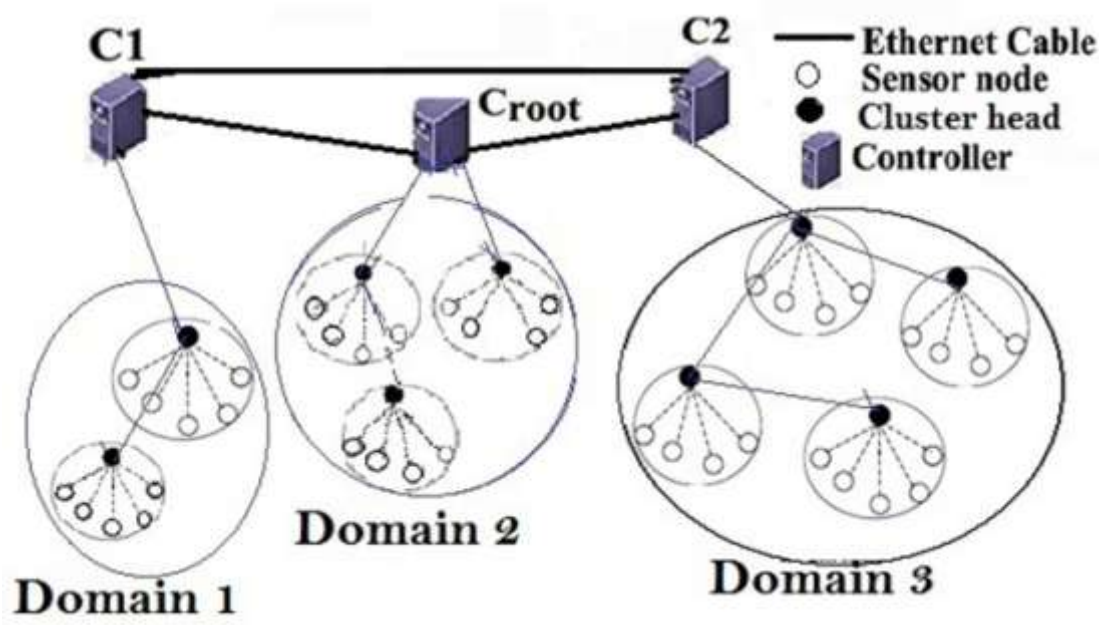

Figure 4: Proposed Network Structure

At the beginning, clustering the nodes is done by applying the k-way spectral clustering algorithm [100] which considers the nodes' spatial positions. Afterward, we applied metaheuristic algorithms (GA, GRASP) to solve the CPP by optimizing various system constraints defined by the fitness function. The fitness function of the mentioned algorithms is a multi-objective function aiming at optimizing various network performance metrics, specifically maximizing the network connectivity, balancing the load, minimizing the network latency, and maximizing network lifetime. In our proposed scheme, we assumed the followings:

- The root controller (C_{root}), placed at the sink, is failure-free.
- Sensor nodes are randomly distributed
- Communication link is shared between control and data planes
- The network topology is known by the root controller
- The forwarding tables consist of disjoint paths to ensure reliability in the presence of link or node failure.
- SDN controllers have the same characteristics.

3.4 System Description

As shown in Figure 4, the model can be considered as an undirected graph G(V, E) with sensor nodes representing the vertices V, and set of links E. The model incorporates several controllers, i =1,2,…; one is the root controller, and the rest are optimally selected using GA and GRASP algorithms from the network's cluster heads {1,…,N}. Hence, each controller i is associated with several cluster heads n_i, where n_i is a subset of N. A chromosome consists of a number of genes where each gene represents a cluster head. A population is a collection of K chromosomes where the detailed description of the population is explained in section 3.5.1.2. For each chromosome in K, k random cluster heads are selected, where $k \subset n_i$, and the fitness function is calculated based on the following four objectives.

• Maximizing the network connectivity: The number of flow messages implicitly reflects the strength of network connectivity. The number of flow messages arriving at controller i from one of its cluster heads $j \epsilon n_i$, is denoted by f_{ji}. An important note to be mentioned is even in the presence of faulty paths or nodes, the transmission of messages is done via alternative paths to reach the destination. For this, we have considered the use of spectral clustering which partitions the network into disjoint clusters. Equation (1) shows the maximum average flow among all the K chromosomes of the randomly selected k controllers, where n_k is the number of cluster heads associated with the related controller.

$$f1=\max_K \sum_{i=1}^{k} \sum_{j=1}^{n_i} \frac{f_{ji}}{n_i} \quad (1)$$

• Balancing the controller's load: Balancing the load among controllers avoids the presence of overwhelming controller(s) which negatively affects the network delay

due to the increase in the controller's response time. When using the in-band scheme, as in our case, balancing the load among controllers is an important aim to be achieved to avoid unnecessary delays. The load of a controller i , denoted by $Load_i$, is the sum of all the successfully flow messages issued by the controller's associated cluster heads jϵ n_i, (i.e. f_{ji}), and neighboring cluster heads j'ϵ n_b of the neighboring controller b, denoted by $f_{j'i}$. Therefore, a message is successfully received if a node j issuing the flow is nonfaulty, denoted by ft_j, as shown by equation (2), and the path between cluster head j and controller i exists, denoted by p_{ji}, as shown by equation (3). Hence, the load of controller i can be shown as given in equation (4).

$$p_{ji}=\begin{cases}1000 & \text{a path exists between a cluster head j, and a controller, i}\\ 1 & \text{otherwise}\end{cases} \quad (2)$$

$$ft_j=\begin{cases}-1 & \text{if cluster head j is faulty}\\ 1 & \text{otherwise}\end{cases} \quad (3)$$

$$Load_i = \sum_{j=1}^{n_i} f_{ji} \times ft_j \times p_{ji} + \sum_{j'=1}^{n_b} f_{j'i} \times ft_{j'} \times p_{j'i} \quad (4)$$

Equation (5) shows the minimum load among all the K chromosomes of the randomly selected k controllers.

$$f2= \min_K(\sum_{i=1}^{k} Load_i) \quad (5)$$

• Minimizing the network delay: The network delay, denoted by D_{Total}, is the sum of transmission delay, propagation delay, and the queuing delay of a controller in each chromosome. The transmission delay is the time taken to push all the packet's bits into the link, and is given by L/B, where L is the packet size, and B is the bandwidth. We neglected the transmission delay in our case as it is very small compared to the other delays. The propagation delay is the time needed for a packet to reach the destination, and the queuing delay represents the waiting time of a packet in the controller's buffer where each controller's buffer is modeled as an M/M/1 queuing system [60]. The service rate is denoted as μ, and $\lambda_{f_{ji}}$ is the arrival rate of requests

from cluster head *j* to controller *i*. The distance between a cluster head *j* and a controller *i* is denoted by d_{ji}, and the speed of light is denoted by *c*. Therefore, the aim is to minimize the maximum worst-case latency, given by equation (6), which must be bounded to a given threshold, denoted by $T_{threshold}$ to avoid unnecessary delays.

$$f3=D_{Total}= \max_K \sum_{i=1}^{k} \sum_{j=1}^{n_i} \left(\frac{d_{ji}}{c} + \sum_{l=1}^{f_{ji}} \frac{1}{\mu-\lambda_l}\right)$$

$$\text{s. t } D_{Total} <= T_{threshold} \quad (6)$$

• Maximizing the network lifetime: The network lifetime represents the total number of alive nodes existing at the end of the simulation. A node *j* belonging to controller *i* dies when the current energy, denoted by $E_{j,}$ falls below an energy threshold, denoted by $E_{threshold}$. The sensor current energy, E_j, is the difference between the initial energy, denoted by $E_{(ini)j}$ and the total consumption energy, denoted by $E_{jconsumption}$. Equations (7), (8), and (9) provide the calculation details for energy consumption for transmitting *L* bits, receiving *L* bits, and total energy consumption. The energy required for a node *j* to transmit *L* bits at a distance d_j is denoted by ET_j and given by equation (7), where E_{elec} is the energy consumption of node transceiver circuit for receiving or transmitting one-bit data, E_{fs} and E_{amp} are power consumption coefficients needed for power amplification in the free channel and multi-path fading channel respectively, and d_0 denotes the distance threshold to decide which radio model is used. The energy required for a node *j* to receive *L* bits is denoted by ER_j and given by equation (8). The total node's energy consumption is the sum of the transmission and reception energies as shown by equation (9).

$$ET_j = \begin{cases} E_{elec} * L + L * {d_j}^2 * E_{fs}, & d \leq d_0 \\ E_{elec} * L + L * {d_j}^4 * E_{amp}, & d > d_0 \end{cases} \quad (7)$$

$$ER_j = E_{elec} * L \quad (8)$$

$$E_{jconsumption} = ET_j + ER_j \quad (9)$$

Therefore, the current energy of a node *j* is given by equation (10).

$$E_j = \quad E_{(ini)j} - E_{jconsumption} \quad (10)$$

Equation (11) shows the link efficiency of the corresponding path, denoted by $linkeff_{ji}$, which must be above a specific threshold, $link_{effthreshold}$. The link efficiency increases the network energy efficiency by considering the consumption energy of the cluster head when sending *L* bits to controllers. It also considers the network bandwidth denoted by B, and the signal-to-noise ratio denoted as SNR.

$$linkeff_{ji} = (B\times \log_2(1+SNR))/ E_{jconsumption} \quad (11)$$

Equation (12) shows the maximum network lifetime among all the K chromosomes of the randomly selected k controllers by avoiding the consideration of faulty cluster heads in the system.

$$f4 = \max_K \sum_{i=1}^{k} \sum_{j=1}^{n_i} \sum_{l=1}^{j} E_l \times linkeff_{ji} \times p_{ji} \times ft_j) \quad (12)$$
$$s.t.\, E_l > E_{threshold}, linkeff_{ji} \geq link_{effthreshold}$$

The fitness function's weight values are given by the scalarization method [66] which is capable of creating a single solution. In particular, we have used the Rank Order Centroid (ROC) weights. Thus, the final fitness function denoted by $\hat{F}$ and given by equation (13) shows the maximum fitness value among all the K chromosomes of the randomly selected k controllers:

$$\hat{F} = \max_K(\omega_1 f1 + \omega_2 f2 + \omega_3 (f3)^{-1} + \omega_4 f4) \quad (13)$$
$$s.t\ \omega_1 + \omega_2 + \omega_3 + \omega_4 = 1$$

The fitness function consists of four objectives, and the ROC [66] weight values are given by equation (14).

$$\omega_i = {}^{1}/_{n_o} \sum_{k=i}^{n_o} 1/k \quad (14)$$

Where n_o is the number of objectives which is 4 in our case. Using equation (14), the weight values are: ω_1=25/48, ω_2=13/48, ω_3=7/48, and ω_4=3/48. The high weight

value of a function reveals a high priority and vice versa. For example, giving f1 a weight value of 25/48 means that the network connectivity has the highest priority over the rest of the objective functions.

The proposed Fault Tolerance Metaheuristic-Based Scheme (FTMBS) incorporates the use of two metaheuristic algorithms (GA and GRASP) for solving the CPP, and the Balance State System algorithm for reaching the steady-state. A detailed description of these algorithms is in the following section.

3.5 Metaheuristic Algorithms

We have applied metaheuristic algorithms to solve the CPP, specifically, the GA and GRASP algorithms. When a network consists of several controllers, finding their best locations is an NP-hard problem [41]. The only solution for such problems in a reasonable amount of time is by using evolutionary algorithms such as, in our case, the Genetic Algorithm (GA) and the GRASP algorithm.

3.5.1 Genetic Algorithm

Inspired by natural evolution, the core operators of the GA are inheritance, crossover, and mutation. GA optimizes a problem by providing near-optimal solutions [49]. The evolution starts with a population of randomly generated chromosomes and happens in generations. In each generation, the fitness of every chromosome is evaluated from the population where multiple chromosomes are selected from the current population using the tournament selection which selects the chromosomes with the highest fitness value. Then, the partially matched crossover and mutation operations are applied to form a new population. This new population is added to the next iteration of the algorithm. The algorithm ends when a maximum number of generations is fulfilled, or an optimal fitness level is reached. The probability of

using the crossover and the mutation operators are 0.8 and 0.2, respectively. The fitness function calculation for each chromosome is provided by Equation (13). The algorithm returns the minimum cost value, the average cost value, and the maximum cost value with the location of the controllers that are the indices of the cluster heads. The pseudo-code of GA [73] is given in Table 2.

Table 2: Pseudo-code of GA

Input: Parameter popsize, crossover probability p_c, mutation probability p_m, maximum iteration iter_max, k is number of controllers
Output: The controllers' locations
1: initialize popsize individuals;
2: check feasibility of each individual;
3: WHILE number of generations <= iter_max do
4: For i = 1 to popsize do
5: Ƒ (i) ← φ
6: Select k controllers from cluster heads, S={1,…,i} is set of cluster head ids
7: For j = 1 to k do
8: Apply the k-mean method to assign every cluster head to controller
9: end for
10: calculate the fitness value Ƒ of each chromosome given in equation 13
11: Order the population based on evaluation value;
12: Perform the Tournament selection process;
13: Apply the partially matched crossover operator
14: Apply the mutation operator
15: Update the population for the next generation;
16: min_cost ←min(Ƒ)
17: avg_cost←average(Ƒ)
18: [max_cost, location] ←max(Ƒ)
19: end for
20: END WHILE
21: return min_cost, max_cost, avg_cost, S

3.5.1.1 Chromosome Definition

A chromosome is a collection of genes of length *N*; in our case, it is a collection of cluster heads found in the given network. Therefore, a chromosome consists of sequences of positive integers that represent the IDs of cluster heads. The variable

length of the chromosome is represented as the total number of cluster heads, *N*, in the network.

3.5.1.2 Population Representation

A population is a collection of chromosomes of population size, popsize, where the initial population consists of several chromosomes with the possibility of having faulty cluster head(s). For each chromosome of the population, a random selection of *k* controllers from *N* cluster heads is performed. Then, the k-mean method to assign the rest of the cluster heads among these controllers is applied. Then the objective function for each chromosome, F is calculated. Then, the selection of two chromosomes, having the lowest fitness value to recover the faulty nodes, is based on tournament selection for finding better chromosomes for fault tolerance in the network. Then, crossover and mutation are applied. The algorithm returns the best solution containing the IDs of the cluster heads for solving the CPP. Figure 5 shows an example of chromosome representation for a given network of several cluster heads, *N*, with a possible solution marked in blue. Each gene value provides the cluster head identification number (ID). Therefore, this chromosome contains the controllers' locations; i.e., the bold colored IDs are the cluster heads where the controllers are placed.

popsize	Cluster heads ids									
	1	2	3	4	5	6	7	8	...	N
1	30	20	1	15	100	450	470	100	...	80
2	37	27	8	22	107	457	477	107	...	87
3	44	34	15	29	114	464	484	114	...	94
4	51	41	22	36	121	471	491	121	...	101
5	58	48	29	43	128	478	498	128	...	108
6	65	55	36	50	135	485	505	135	...	115
7	72	62	43	57	142	492	2	142	...	122
8	79	69	50	64	149	499	8	149	...	129
9	86	76	57	71	156	500	3	156	...	136
10	93	83	64	78	163	18	30	163	...	143
11	100	90	**71**	85	**170**	19	37	170	...	**150**
12	107	97	78	92	177	200	100	177	...	20
13	114	104	85	99	184	500	107	184	...	27
14	121	111	92	212	191	500	8	3	...	34
15	128	118	99	219	198	200	40	10	...	41
⋮						⋮				
50	142	132	113	233	212	200	8	24	...	12

Figure 5: Population Representation

3.5.2 Greedy Randomized Adaptive Search Procedure Algorithm

The GRASP algorithm initially proposed for the Operations Research practitioners [55] consists of two related phases; the construction and local search phases. The ending criterion and the elements' quality of the restricted candidate list, denoted by RCL, are the two basic parameters of GRASP. A result consists of the best solution. The construction phase consists of iteratively constructing a feasible solution, one element at a time. Then, the elements in the candidate list are ordered according to a greedy function. GRASP consists of a probabilistic component where the best candidates are randomly chosen. These best candidates are placed in a list, namely RCL. The pseudo-code of GRASP can be found in [55].

3.6 Balance State System (BSS)

Before describing the balance state system's details (BSS), it is necessary to determine the controller's response time threshold since it is a triggering factor for the BSS. At time interval T_n, a controller *i* response time for the request issued by its associated cluster head member j ($j \epsilon n_i$), denoted by $Resp_{ij}$, is calculated by subtracting the time the request has arrived at the controller, denoted by t_{arrive} from

the time the controller issues its response, denoted by $t_{reply,}$ as given by equation(15). The response time of a controller *i* for a request issued by a neighboring cluster head j' ϵ n_b belonging to controller b is denoted by $Resp_{ij'}$. Hence, the average response time of controller *i*, denoted by AvgRi, is the summation of all the response time divided by the load defined in equation (4). The controller's average response time is given by Equation (16):

$$Resp_{ij} = t_{reply} - t_{arrive} \quad (15)$$

$$\text{AvgRi} = \frac{\sum_{j=1}^{n_i} Resp_{ij} + \sum_{j'}^{n_b} Resp_{ijj}}{\text{Load}_i} \quad (16)$$

The BSS incorporates two interrelated algorithms which are executed by the root controller consecutively. The first algorithm is the Balance State Awareness (BSA) algorithm presented in Table 3, and the Balance State Warranty (BSW) algorithm presented in Table 4. The BSA detects the imbalance state, whereas the BSW ensures that the network steady-state is achieved. As stated before, the controller's response time is used as a trigger parameter that the root controller checks to execute the BSA algorithm. The root controller checks if a non-faulty controller's average response time exceeds a certain threshold, denoted by RespT_threshold. The controller ensures the network steady-state by executing the BSS algorithm. A significant note for claiming a controller to be overloaded is by checking if it is not faulty. This condition is essential as a controller could be faulty but not overloaded. This condition is missed in [48] [74]. The extreme point of the changing curve in the first and second-order derivatives, denoted by AvgRi'(Tn) and AvgRi"(Tn) respectively, represents the threshold of the controller's average response time at time Tn. The first and second-order derivatives are given in (17) and (18), respectively.

$$\text{AvgRi'Tn} = \frac{\text{AvgRiT}_n - \text{AvgRiT}_{n-1}}{T_n - T_{n-1}} \quad (17)$$

$$AvgRi''Tn = \frac{AvgRi'Tn - AvgRi'Tn-1}{T_n - T_{n-1}} \quad (18)$$

After recording the average response time of the controller for different load values and finding the first and second derivatives given by equations (17) and (18), we found that the point referring to a value of 2ms is a global maxima point having a second derivative negative and first derivative zero. We recorded the controller's average response time versus load as shown in Figure 6. Accordingly, the controller's load threshold, denoted by $Load_{threshold,}$ is set to 2600 packets. Table 3 illustrates the BSA of the BSS. The algorithm's inputs are the average response time of all controllers found in the system and the obtained threshold. Two sets denoted as OL_C for overloaded controllers and LL_C for low-loaded controllers are empty at the beginning. The root controller checks if all the controllers' average response time satisfies the condition of overload; i.e., the controller's average response time is above the obtained threshold, and the controller is not faulty, then the controller is added to the OL_C set, else, it is added to LL_C set. However, if the controller is not overloaded (i.e. faulty), the root controller executes the third algorithm, namely the Load distribution of the Faulty Controller denoted by LFC. The LFC algorithm distributes the faulty controller's load among the non-faulty and non-overloaded controllers, ensuring the network functions smoothly with no delays. Then, the root controller applies the metaheuristic algorithms to find another location among the nonfaulty cluster heads for the faulty controller.

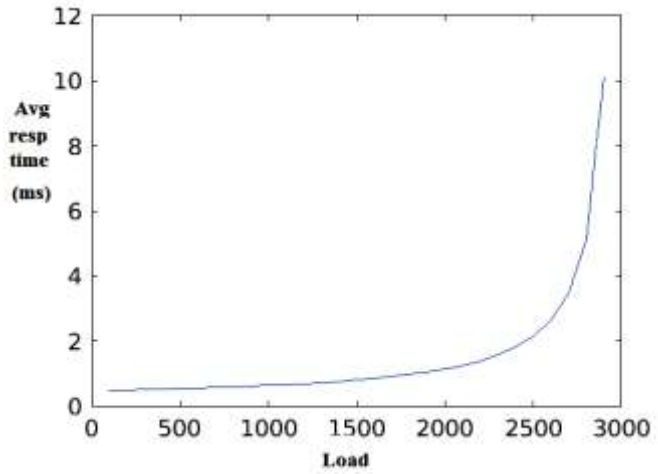

Figure 6: Controller's Average Response Time vs. Load

When the BSA is triggered, the two produced sets (OL_C and LL_C) are used in the BSW algorithm to guarantee that no overloaded controller(s) exists. The BSW algorithm handles the load migration process efficiently between controllers. The following steps describe the BSW algorithm shown in Table 4.

Step 1: find the maximum overloaded controller denoted by C_O in line 3. Step 2: find the heaviest overloaded cluster head denoted by *j* belonging to the maximum overloaded controller and a nonfaulty node having its flag is false to avoid choosing the same node in line 4. Step3: find the minimum loaded controller denoted by C_L as a recipient controller after migrating *j* to it in line 5. However, before the migration process occurs, the root controller makes sure that the recipient controller's load is below the given threshold, denoted by $Load_{threshold.}$ Then, the migration process is done to P <j,C_L> in line 9; otherwise, the root controller flags this j and continues with the next heavy cluster head of an overloaded controller in line 7. Table 4 illustrates the BSW migration process, which ensures no overloaded controller exists. The condition of no overloaded controller is satisfied if the overall load of the recipient low-loaded controller CL is below the load threshold; then, the migration

process is done. Otherwise, the heavy cluster head is flagged, and the algorithm continues with the next heavy cluster head of the overloaded controller.

The third algorithm is the Load Distribution of Faulty Controller (LFC), illustrated in Table 5, which distributes the load of cluster heads of a faulty controller in a balanced way among other controllers considering not to end up having an overloaded controller. We specify the threshold load that is the border for not having overloaded controllers. The algorithm starts with input information such as the load of cluster heads of the faulty controller, load information of the other controllers, and the load threshold. First, the set A of load info of CHs is sorted in decreasing order, starting from the heaviest CH, and sort the controllers, starting from the lightest one. An essential condition for the addition of CH to the controller is that, the controller's load should not exceed a threshold denoted by $Load_{threshold}$. If this condition is satisfied, the CH is added to the current controller. Otherwise, it is added to the second lightest controller only if the receipt controller's total load is within the load threshold. Otherwise, the cluster head is flagged. The algorithm then continues with a non-flagged heavy cluster head to determine which controller to choose for migration. The algorithm stops when cluster heads of faulty controllers are distributed among other controllers; i.e., no CHs whose flag=false exist.

Table 3: Balance State Awareness (BSA)

Input: A = { AvgRi, iϵ{1,..,k}, k=3 or 5,RespT_threshold
Output: OL_C, LL_C
1: initialize controller set OL_C = { } & LL_C = { }
2: let i be the serial number of the controller
3: For i = 1 to k do
4: select $AvgR_i$ from A , s. t. fault(i) =false
5: if AvgRi > RespT_threshold then
6: add i to OL_C
7: else
8: add i to LL_C
9: end if
10: if fault(i)==True then execute LFC
11: end if
12: end For
13: return OL_C, LL_C

Table 4: Balance State Warranty (BSW)

Input: OL_C, LL_C, load information j cluster heads of overloaded controller Co, f_{jCo}, $Load_{threshold}$, C_{root}
Output: P: CH migration actions set
1: initialize node shifting set P ={ }, flag(all CHs,j)=false, Fault(all CHs,j)=false
2: while (OL_C isNotEmpty & flag==false) do
3: $C_O = Max_{Co \epsilon OL\_C}\{Load_{Co}\}$; find maximum loaded controller Co
4: j = Max $\{f_{jCo}\}$s.t fault(j)==false and flag(j)==false; find cluster head of maximum load
5:$C_L = Min_{CL \epsilon LL\_C}\{Load_{CL} + f_{jCo} <= Load_{threshold}\}$
6: If C_L==NULL then Flag(j)=true
7: Go to step 2
8: else
9: add< j;C_L> to P
10: Remove C_O from OL_C
11: Remove C_L from LL_C
12: end if
13: end while
14: return P

Table 5: Load distribution of Faulty Controller (LFC)

Input: Load information of all cluster heads j ϵ n_x of faulty controller x, f_{jx}, load of nonfaulty controllers at time Tn, $Load_{threshold}$, number of nonfaulty controllers k, number of cluster heads n_x of faulty controller x.
Output: P: CH migration actions set
1: initialize A set of loads of CHs of faulty controller, B is set of loads of nonfaulty controllers, Flag all cluster heads jϵ (1,…,n_x)=false
2: Sort in decreasing order set A={f_{1x}, f_{2x},…., f_{Nx}}
3: Sort B in increasing order B={ $Load_1$, $Load_2$,.., $Load_k$}
4: while A IsNotEmpty do
5: for i=1 to k do
6: for j=1 to n_x do
7:If ($Load_i$ + flag(f_{jx}=false)<=$Load_{threshold}$) then
8: Add <j,i> to P
9: Update $Load_i$ and the set B
10: Remove load information of j from A
11: Goto 2
12: else
13: Flag(j)==True
14: Goto 4
15: end if
16: end for
17: end for
18:end while
19: return P
20: Execute the metaheuristic algorithms to solve CPP

First, three controllers were considered, and various network performance metrics were recorded under the execution of the proposed FTMBS scheme. Afterward, we considered five controllers, recorded the same network performance metrics, compared them using three controllers, and suggested the appropriate number of controllers.

The FTMBS flowchart is shown in Figure 7. We have 500 sensor nodes randomly distributed in a specific field area. We have used the K-way Spectral Clustering Technique [100] to partition the network into several disjoint clusters. The cluster heads are selected based on their current energy; i.e., the current energy should be

above a given threshold, denoted by $E_{threshold}$. The run time is set to 300 seconds with an arrival rate of 100 packets per second (pps) and service rate of 4000 pps and increasing every 10 seconds by 100 packets till the arrival rate is 3000 pps. The root controller executes the BSS algorithm to keep the network in a steady state. Whenever faulty controller(s) exists during a time interval of 10 seconds and less than the simulation time, the root controller will execute the GA or GRASP to solve the CPP. Applying the genetic algorithm to solve the CPP satisfying the mentioned constraints, simulations for different fault percentages and network performance metrics are recorded accordingly. Then, the root controller solves the CPP by running the GRASP algorithm. Finally, network performance metrics comparisons between GA and GRASP are carried out when three and five controllers are present with and without the BSS algorithm.

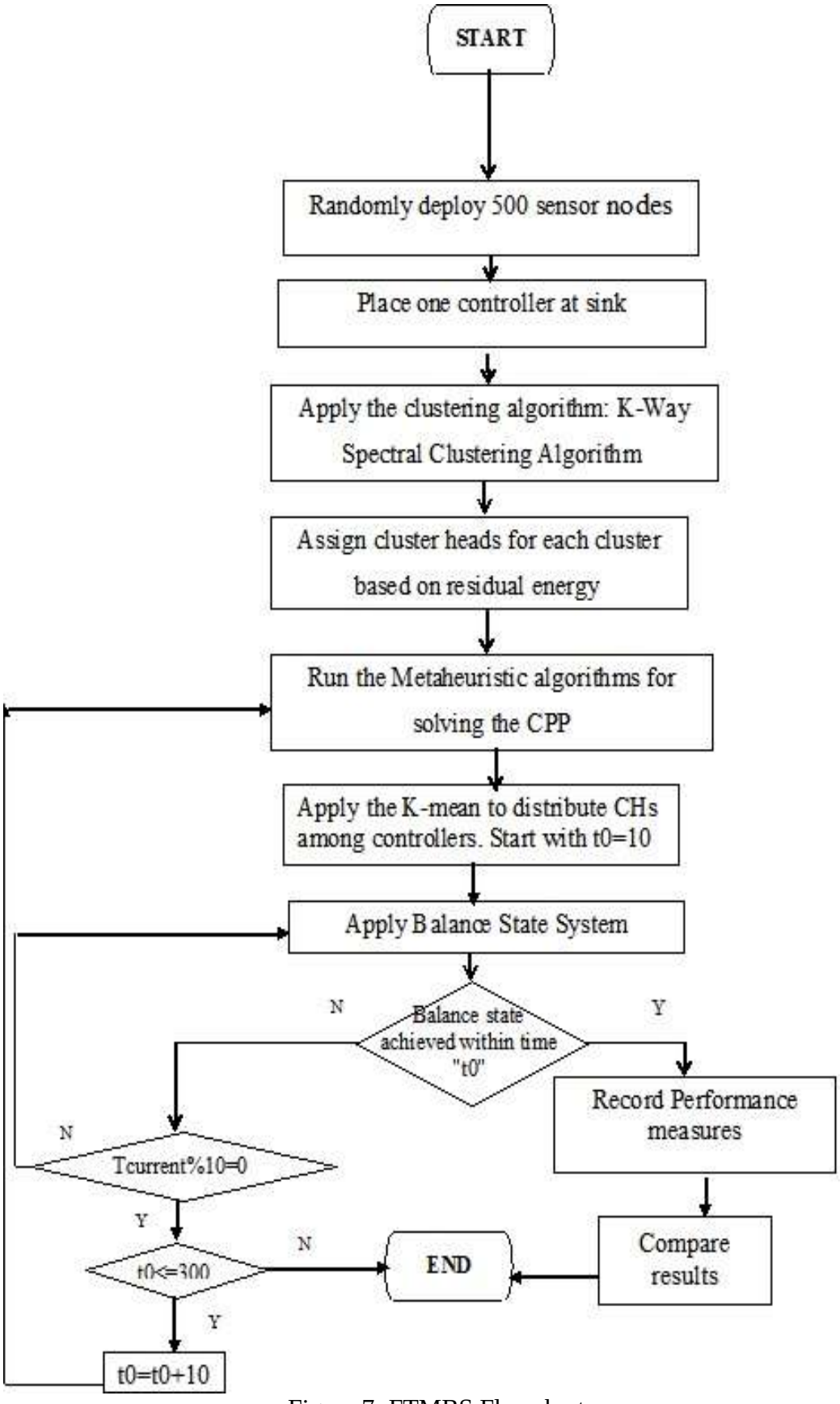

Figure 7: FTMBS Flowchart

Chapter 4

PERFORMANCE EVALUATION

We have used MATLAB 2019 to run our proposed FTMBS scheme under various faulty node percentages. Table 6 presents the simulation parameters [89]. We have run our scheme 50 times for each fault percentage and took the average to achieve a 95% confidence interval. As described in Chapter 3, we have determined the controller's average response time threshold, in our case 2 ms, which is the triggering factor used for determining overloaded controllers. It is also used as an input parameter for the balance state system. We have defined the required average percentage of improvements (API) ahead of time, 40% and above for latency and successful packets received, 20% and above for controller average load and controller's average response time, and 15% and above for the number of alive nodes. Finally, we have included the confidence interval estimation for the latency values under the GA for three controllers. The confidence interval estimation is provided in Appendix B. We have also compared the worst-case latency of FTMBS with a clustering-based network partition algorithm (CNPA) [89], where the authors place the SDN controllers randomly on the centroid of the clusters found in the network.

Table 6: Simulation Parameters

Parameter	**Value**
Field area	200x200 m^2
No.of sensor nodes, V	500
Crossover probability	0.8
Mutation probability	0.2
Population size, popsize	50
Number of runs, iter_max	50
Sensor Energy	2 Joules
Energy Threshold, $E_{threshold}$	0.05 Joules
Sensing Energy	50nJ/Bit
Energy required in sending or receiving 1bit(E_{elec})	50nJ/Bit
Packet arrival rate	200packets/s-3000packets/s
Packet length, L	30 bytes
Service rate, μ	4000packets/s
Bandwidth, B	2Mbps
E_{fs}	10pJ/bit/m2
E_{amp}	0.0013pJ/bit/m4
d_0	87m
Time interval	10 seconds
Time period	300 secs / 5mins
RespT_threshold	2ms
$Load_{threshold}$	2600 packets
$link_{effthreshold}$	0.2mbps
Node Fault probability	[0,1,5,10,20,50]%
Signal to noise ratio, SNR	40 dB
Latency Threshold, $T_{threshold}$	1.2 ms

Analysis of the results is carried based on the following network performance metrics:

4.1 Worst-Case Latency

We have conducted different simulations for analyzing the worst-case latency given by equation (6) to examine the performance of the proposed FTMBS. First, we recorded the latency when no heuristic algorithms are executed. We randomly placed the required number of controllers on cluster heads and recorded the latency. Then, we applied the cluster-based network partition algorithm (CNPA) [89] that considers no heuristic algorithm to solve the CPP. On the contrary, authors have partitioned the

network into k given clusters by finding the centroids and allocating the nodes among these centroids, then placing the controllers at the centroids. We have compared the worst-case latency with these algorithms.

4.1.1 Worst-Case Latency Analysis for FTMBS

As described in Chapter 3, we have recorded the latency results when running the GA and GRASP algorithms with and without the BSS for three and five controllers. The BSS has a positive impact on network latency under GA in the presence of faulty nodes for three and five controllers as shown in Figure 8 and Figure 9, respectively. The same impact under the GRASP algorithm can also be observed as shown in Figure 10, and Figure 11, respectively. We can see that the latency has been decreased when using five controllers instead of three controllers for both GA and GRASP algorithms. Taking into account the required API mentioned at the beginning of this chapter, we imply using three controllers instead of five controllers despite the marginal improvement in latency achieved with the five controllers case.

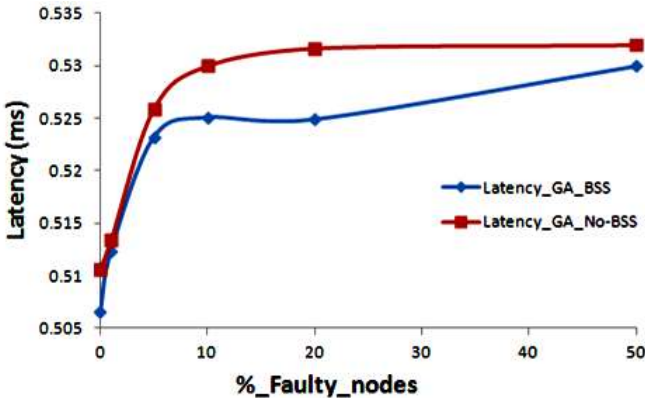

Figure 8: Network Latency for 3 Controllers under GA

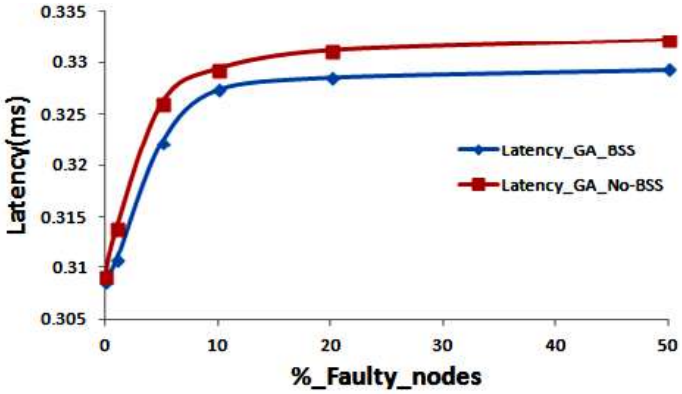

Figure 9: Network Latency for 5 Controllers under GA

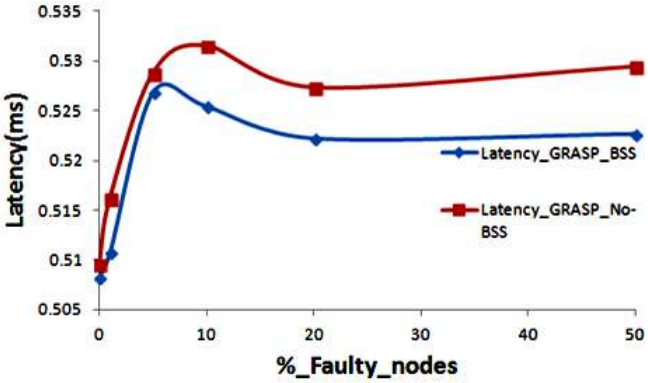

Figure 10: Network Latency for 3 Controllers under GRASP

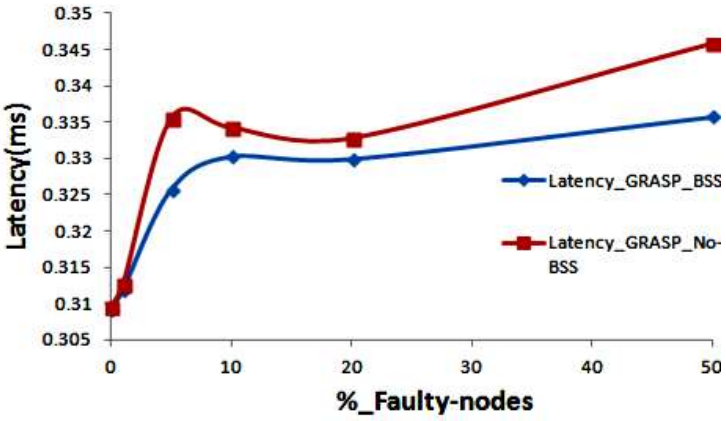

Figure 11: Network Latency for 5 controllers under GRASP

4.1.2 Worst-Case Latency Analysis for Different Algorithms

First, we have considered three controllers to be implemented on the cluster heads. The first method randomly places the three controllers on the cluster heads found in

the network after applying the k-spectral clustering method. The second method applies the CNPA proposed by Wang et al. [89], where the network is clustered into k clusters, k value varies as [1-6]. In our case, we choose k=3. The algorithm randomly chooses a node to be the center of the network. It tries to find the actual centroid by calculating the sum of worst-case latency between pairs of nodes and finds the node of minimum sum latency to be the first centroid. The node with the maximum sum of worst-case latency is selected as the second centroid. Then, the algorithm allocates the nodes to these two centroids based on minimum latency. Once a node is allocated to one of the centroids, recalculation for the centroid is done, and this continues till the network is divided into k clusters. We have considered CNPA and random algorithms with and without the BSS algorithm to analyze the impact of BSS on latency. Figure 12 shows the worst-case latency of the algorithms mentioned above compared to the proposed FTMBS algorithm with and without the BSS under the GA. In the random method, the network is clustered using k-spectral clustering, which gives the optimal number of disjoint clusters. When a network is well clustered, the latency is decreased. On the other hand, in CNPA, the network is clustered into three clusters, and the controllers are placed at the centroids of these clusters. This explains the cause behind having the latency of the random method less than that of CNPA. Also, the proposed FTMBS under the GA has achieved the least worst-case latency compared to the random method and CNPA algorithm. In the proposed FTMBS, the placement of controllers is formulated as a multi-objective function to optimize the worst-case latency, connectivity, load balance among the controllers, and network lifetime. When the BSS algorithm is executed on FTMBS, the nodes selected as cluster heads deplete energy during simulation; therefore, the controller selects another node to be cluster head. In

addition, the location of each controller is not fixed. On the contrary, the root controller executes the metaheuristic algorithms to optimally select the controllers once faulty controllers are found. However, in CNPA, the selected nodes as centroid for every cluster after finding the k clusters do not change during simulation, which is a drawback as a sensor node depletes energy due to transmitting data. This drawback is negatively reflected in the network connectivity, reliability, and overall network performance. Hence, the FTMBS algorithm shows its superiority, even in the presence of faulty nodes, over CNPA and random algorithms. In addition, the BSS algorithm shows a positive impact on reducing the latency that is affected by overloaded or faulty controller(s). As shown in Figure 8, the BSS algorithm has 8% and 10% improvements in latency over random and CNPA algorithms, respectively. Also, the FTMBS with the BSS algorithm has 11% and 14% improvements over random and CNPA algorithms, respectively. Hence, the superiority of the proposed FTMBS is obvious.

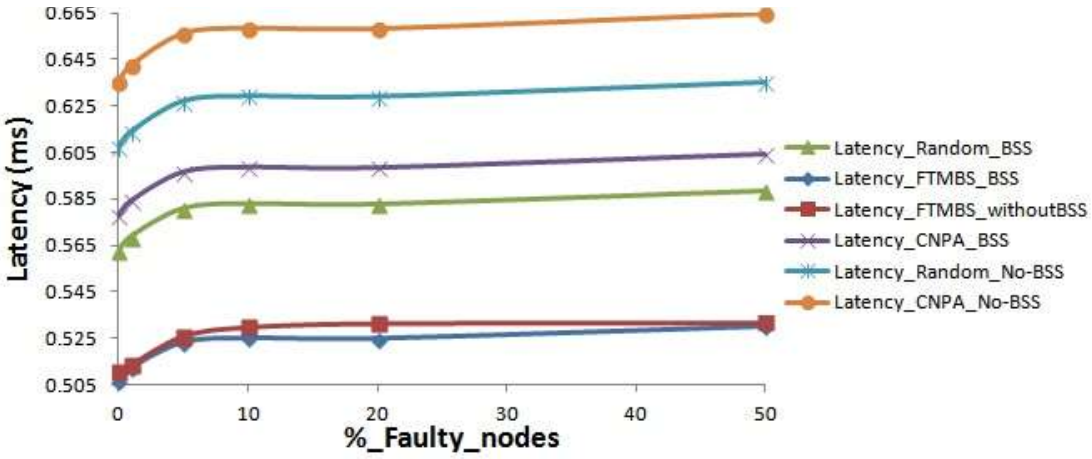

Figure 12: Latency Analysis for Different Algorithms

4.2 Network Lifetime

Recall that network lifetime is defined as the total number of alive nodes existing at the end of the simulation given by equation (12). The more the network energy is saved, the more alive nodes will exist. Table 7 and Table 8 show the total number of alive nodes under GA and GRASP algorithms with and without the BSS algorithm when three and five controllers are used, respectively. As shown in Table 7 and Table 8, the number of alive nodes for both algorithms with BSS is higher than without BSS. Also, the number of alive nodes under the GRASP algorithm is more than that under the GA algorithm. However, the BSS has more effect on improving the number of alive nodes under the GA algorithm than the GRASP algorithm. Using five controllers, the number of alive nodes is slightly more than when using three controllers under the GA algorithm. The API for GA and GRASP algorithms with and without BSS algorithm are 13% and 11 % when using three controllers compared to 24% and 20% when using five controllers, respectively.

Table 7: Network Lifetime for 3 Controllers

Fault %	**Number of Alive Nodes for 3 Controllers under Different Fault Percentages**					
	GA	**GRASP**	**GA**	**GRASP**	**% improvement**	
	with BSS		**without BSS**		**GA**	**GRASP**
0	304	419	303	411	0.3	1.9
1	235	252	205	250	15	1
5	196	237	173	234	13	1
10	170	203	145	189	17	7
20	149	190	137	146	9	30
50	46	57	37	46	24	23

Table 8: Network Lifetime for 5 Controllers

Fault %	Number of Alive nodes for 5 Controllers under Different Fault Percentages					
	GA	GRASP	GA	GRASP	% improvement	
	with BSS		without BSS		GA	GRASP
0	413	421	314	382	31	10
1	243	254	198	246	23	2
5	198	245	189	204	4	20
10	189	241	154	200	23	21
20	181	198	148	168	22	18
50	48	63	34	42	41	50

4.3 Execution Time of Balance State System

We used the MATLAB utilities to determine the execution time of the BSS. As seen in Table 9, the execution time of BSS is more when using five controllers than when using three controllers under both the GA and GRASP algorithms. Since connectivity is improved when using five controllers, the execution time needed to achieve this connectivity improvement increases. However, in GRASP, the execution time is less than that in GA. The execution time improvement percentages, denoted as % fastness, in GRASP is better or faster when using three controllers than when using five controllers, as shown in Figure 13.

Table 9: Execution Time of BSS (sec)

Fault %	Exec.Time_BSS (sec) for 3 Controllers.			Exec. Time_BSS (sec) for 5 Controllers.		
	GA	GRASP	%fastness	GA	GRASP	%fastness
0	0.08	0.06	35.05	0.11	0.09	24.15
1	0.12	0.09	32.19	0.21	0.17	23.10
5	0.15	0.12	27.23	0.24	0.20	21.35
10	0.16	0.12	23.07	0.26	0.22	17.51
20	0.17	0.15	7.36	0.27	0.26	3.62
50	0.29	0.27	7.21	0.38	0.37	2.67

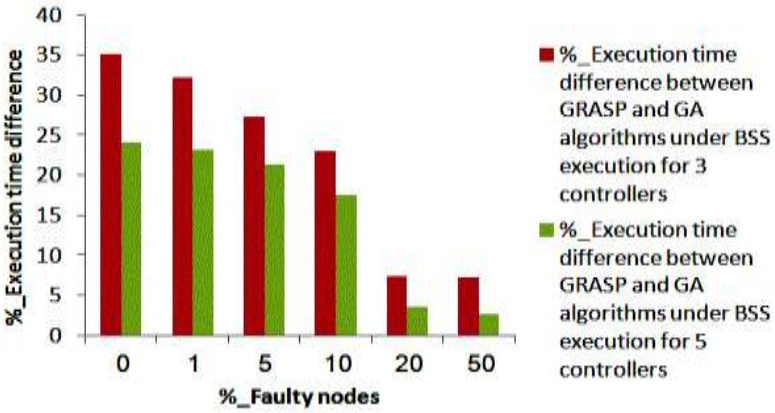

Figure 13: Percentage of Execution Time Difference Comparison

4.4 SDN Controller's Average Response Time

The controller's average response time given by equation (16) is a critical factor which implicitly indicates the presence of overloaded controller(s). Therefore, it is counted as an essential factor for improving the network QoS. Table 10 presents the average response time, measured in nanoseconds (ns) when using three and five controllers under GA and GRASP algorithms with BSS. Since the load of five controllers is more well distributed than three controllers, it is clear that the average response time of five controllers is less than three controllers under both algorithms. Also, the average response time of the controllers under the GRASP algorithm is better than that of the GA algorithm for three and five controllers. However, the percentage of improvement in controllers' response time when using three controllers is much more than that when using five controllers, as shown in Figure 14.

Table 10: Controller's Average Response Time (ns)

Fault %	Avg. Resp. Time (ns) :3 controllers			Avg. Resp.Time (ns):5 controllers		
	GA	GRASP	% improvement	GA	GRASP	% improvement
0	541	438	19	341	338	1
1	548	440	19	348	340	2
5	551	445	19	351	345	2
10	553	450	18	353	350	1
20	554	451	18	354	351	1
50	555	452	18	355	352	1

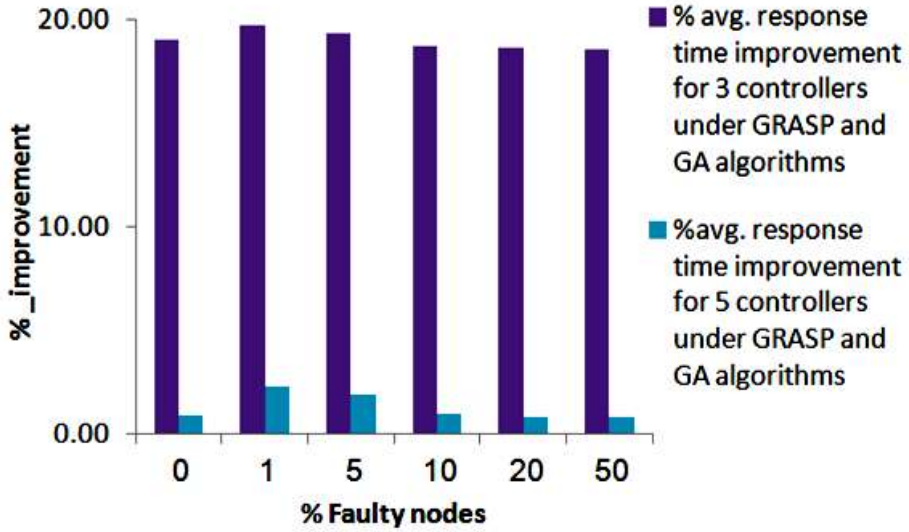

Figure 14: API of Controller's Average Response Time Comparison

4.5 Average Controllers' Load under GA and GRASP Algorithms

We have recorded the average load of controllers given by equation (4) when using three and five controllers. It can be seen that when three and five controllers are present in the network, the controllers' load is well distributed for GA and GRASP algorithms with the BSS algorithm. However, when three controllers are used and less than or equal to 10% of nodes are faulty, the load is improved under GA better than GRASP and vice versa when 20% of nodes are faulty. The obtained results are illustrated in Table 11. When using five controllers and less than or equal to 20% of

faulty nodes are present, the load is improved under GA better than GRASP and vice versa when 50% of nodes are faulty, as illustrated in Table 12. It is worth to notice that, as the percentage of faulty nodes increases, the percentage of load improvement decreases.

Table 11: Average Controllers' Load for 3 Controllers

Fault %	**Average Controllers' Load for 3 Controllers**					
	GA	**GRASP**	**GA**	**GRASP**	**% Load improvement**	
	with BSS		**without BSS**		**GA**	**GRASP**
0	44	42	48	42	6	2
1	76	77	93	84	18	8
5	125	118	140	130	10	9
10	135	132	160	140	15	6
20	161	140	170	149	5	6
50	170	158	188	180	10	12

Table 12: Average Controllers' Load for 5 Controllers

Fault %	**Average Controllers' Load(PKTS) for 5 Controllers**					
	GA	**GRASP**	**GA**	**GRASP**	**% Load improvement**	
	with BSS		**without BSS**		**GA**	**GRASP**
0	34	32	57	52	40	39
1	66	57	87	73	24	21
5	118	113	125	116	5	2
10	128	127	154	144	16	6
20	132	136	165	147	20	7
50	160	153	168	163	5	6

4.6 Percentage of Successfully Received Packets

One of the critical QoS factors is the percentage of successfully received packets. The average number of successfully received packets by controller *i* is the ratio of total flows issued by its associated nodes n_i to its load. Having *k* controllers in the system, the average percentage of successfully received packets in the system, denoted by PR, is given by equation (19).

$$PR=\frac{1}{k}\left(\sum_{i=1}^{k}\sum_{j=1}^{ni}\frac{f_{ji}}{Loadi}\right)\times 100 \quad (19)$$

Figure 15 and Figure 16 show the difference between the percentage of successfully received packets when GA and GRASP algorithms are executed with and without the BSS, respectively. The percentage of packets received when using BSS is more for both algorithms than when no BSS is used. When no faulty nodes are present in the system and five controllers are used, the percentage of successfully received packets is almost 20% better than when three controllers are used. As the percentage of faulty nodes increases, the percentage of successfully received packets decreases as expected.

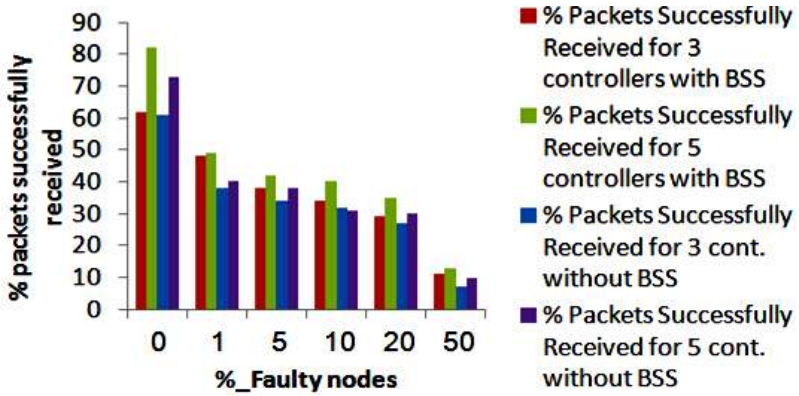

Figure 15: Percentage of Successfully Received Packets under GA and BSS

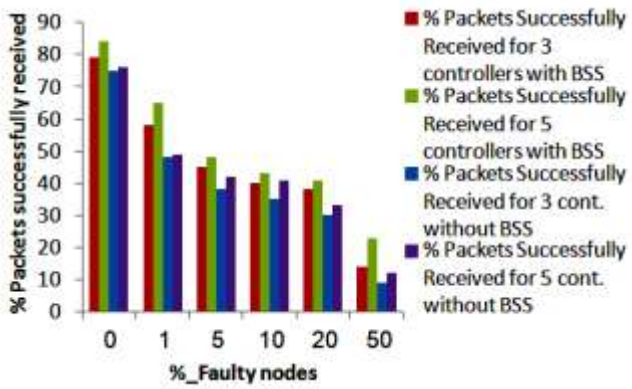

Figure 16: Percentage of Successfully Received Packets under GRASP and BSS

4.7 Complexity Analysis

The main source of the computational complexity in the FTMBS is the metaheuristic algorithm which is in $O(M \cdot popsize^2)$, where M is the number of sub-objectives, and popsize is the population size. To express the overall computational complexity of the proposed scheme, we also have to consider the number of iterations iter_max and the number of cluster heads, K, that are included in the chromosome structure. The complexity of the individual evaluation process is bounded by computing the load of the cluster heads in O(|K|), computing the delay of the graph in O(|K|), computing the network lifetime, which is in $O(|E|^2 \cdot |V|)$. The computational complexity is bounded roughly in $O(iter\_max \cdot (M \cdot popsize^2 + popsize \cdot |E|^2 \cdot |V|. |K|^2))$, which is acceptable for the proposed network structure.

4.8 Discussions

We have recorded the best, average, and worst fitness values on an average of 50 runs when different percentages of faulty nodes exist in the network. It can be seen from Figure 17 that the proposed FTMBS algorithm achieves fault recovery, where for instance, the lowest or worst fitness value when 1% of faulty nodes exist is -0.83×10^6 and gradually increases to reach 3.25×10^6 on an average of 50 runs. It can also be seen that the fitness value is affected by the percentage of faulty nodes.

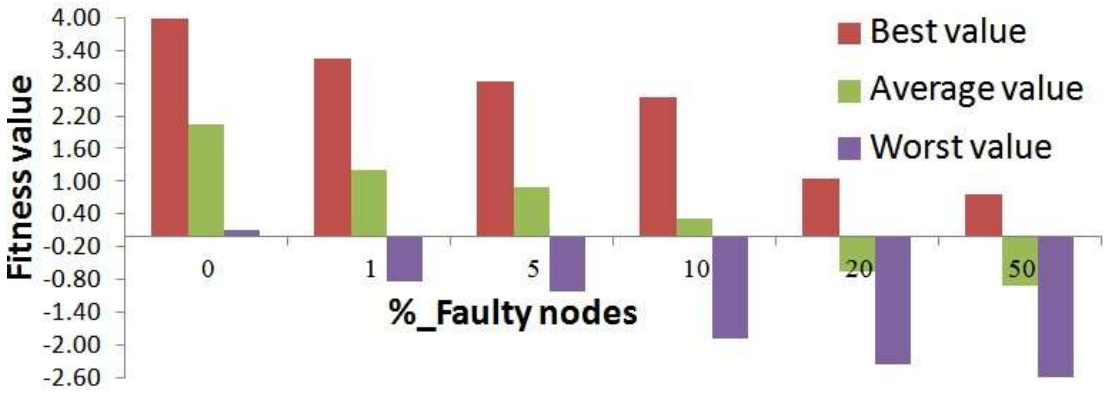

Figure 17: Fitness Value for Different Percentages of Faulty Nodes

The average percentages of network performance improvement of five controllers over three controllers are shown in Table 13. The related performance metrics have been recorded for three and five controllers under different faulty node percentages for both GA and GRASP algorithms with BSS. As seen in Table 13, those improvements vary for different percentages of faulty nodes. Recall that the predefined criteria for selecting five controllers are: 40% and above for latency and successful packets received, 20% and above for controller average load and controller's average response time, and 15% and above for the number of alive nodes. Since the obtained results are 38%, 36%, 17%, 13%, and 11%, we recommend implementing three controllers instead of five controllers.

Table 13: Percentages of Network Performance Improvement of 5 Controllers over 3 Controllers

Fault %	**Latency**		**Avg. Resp. Time**		**Avg. Load**		**No.Alive Nodes**		**Successful Packets Received**	
	GA	**GR**	**GA**	**GR**	**GA**	**GR**	**GA**	**GR**	**GA**	**GR**
0	39	39	37	23	22	24	36	0	32	6
1	39	39	36	23	13	26	3	1	2	12
5	38	38	36	22	5	4	1	3	11	7
10	38	37	36	22	5	4	11	19	18	8
20	37	37	36	22	18	3	21	4	21	8
50	38	36	36	22	6	3	4	11	18	28
API	**38**	**38**	**17**	**17**	**13**	**6**	**11**	**10**	**36**	**22**

4.9 Verification using NSGA-II

In our study, a heuristic approach called nondominated sorting genetic algorithm NSGA-II [52], [104] has been adopted to verify the goodness of the proposed FTMBS scheme for the controller placement problem. Goodness here is referred by how accurate the GA solutions are approximated to Pareto optimal front obtained by NSGA-II. NSGA-II is a population-based procedure used in recent studies for its efficiency in finding the Pareto frontier by applying two main approaches: first, nondominated sorting of the population via elitism and second, obtaining diversification via the crowding distance approach. We have used MATLAB 2019 and manipulated the NSGA-II [70] code to fit our multi-objective optimization problem.

The primary tool for viewing solutions in any multi-objective optimization problem is using the scatter plot [104]. However, this approach analyses only two objectives at a time by focusing on the quality of a set of solutions, relations, and distributions. Unfortunately, the scatter plot is applicable in 2D or 3D Cartesian coordinate space, which is difficult to apply to this study as our multi-objective optimization problem consists of four dimensions. Therefore, an alternative to view data in four or more dimensions is by recording the Euclidean distance between GA solutions and the Pareto-optimal solutions obtained by NSGA-II. The minimum value refers to which optimization criteria the set of solutions belongs to. Table 14 illustrates the Euclidean distance obtained for GA solutions for different weight values according to the optimization target. The first column is for optimizing the connectivity by giving highest weight value for ω_1; the second column is for optimizing the load balance by giving highest weight value for ω_2; the third column is for optimizing the worst-case

latency by giving highest weight value for ω_3; and the fourth column is for optimizing the network lifetime by giving highest weight value for ω_4. Indeed, by giving the highest weight value to connectivity as shown in column 1, the Euclidean distance is the smallest for f1, referring to connectivity optimization compared to the rest of the objectives. The same can be seen for optimizing the load balance by giving ω_2 the highest value, and so on. We can conclude that a good approximation of our solutions to the Pareto frontier has been achieved.

Table 14: Euclidean Distance for GA Solutions

Pareto optimal NSGA-II	**Euclidean Distance Between GA Solutions and NSGA_II Solutions**			
	ω_1=25/48, ω_2=13/48, ω_3=7/48, ω_4=3/48	**ω_1=13/48, ω_2=25/48, ω_3=7/48, ω_4=3/48**	**ω_1=13/48, ω_2=7/48, ω_3=25/48, ω_4=3/48**	**ω_1=13/48, ω_2=3/48, ω_3=7/48, ω_4=25/48**
f1	0.01	2.4	3	3.2
f2	2	0	4.5	7
f3	4	5.8	0.1	5
f4	7	6.9	7.2	0.15

Chapter 5

THE ALBATROSS SCHEME

This chapter presents the ALBATROSS scheme which focuses on improving the wireless sensor network lifetime by adopting the flying technique of the albatross bird in the cluster head selection process. In fact, this scheme is a modification of the FTMBS scheme presented in Chapter 3. However, in FTMBS, nodes are eligible to be cluster heads if the nodes' energies are above a given threshold. Opposing to the cluster head selection process in FTMBS, here, we propose a new cluster head selection process integrated with the dynamic energy soaring scheme which is adopted from nature. Thus, we named the whole scheme as the ALBATROSS scheme. The details of the main component of this scheme, and it’s performance analysis are presented below.

5.1 Dynamic Energy Soaring Scheme (DESS)

The albatross bird is a clever creature that uses the windshields to avoid exhausting itself energy when flapping its wings, and as a result, is able to travel long distances. Since the wings' excessive flapping depletes the bird's energy, the albatross bird soars between different pressure windshields to keep itself lifted. The albatross bird's flying or soaring technique has inspired researchers at the Massachusetts Institute of Technology to develop a new wind and energy harvesting model by adopting the albatross dynamic soaring flying technique. The model focuses on designing energy-efficient wind-propelled drones and gliders to monitor the remote regions for long-duration, long-range under various wind conditions [105]. The dynamic soaring

technique saves the bird's energy and a lot of effort, allowing the albatross bird to travel far distances in a single day with minimum flaps of its wings [105]. For centuries, scientists have noted that these specific kinds of birds make use of different windshields by soaring and diving to keep themselves flying for hours, precisely just above the ocean surface as if riding a sidewinding rollercoaster [105], [106]. Figure 18 shows the heart rate of the albatross bird during different postures [106]. It shows that the albatross bird's heart rate during flying posture falls almost to near basal levels. That is, it falls to the resting posture. However, researchers in [105] have revealed that the birds adaptively fly by shifting among the high-pressure windshields and low-pressure windshields by an average angle of 60 degrees, contradicting the scientists' claim of flying in 180 degrees or half-circle.

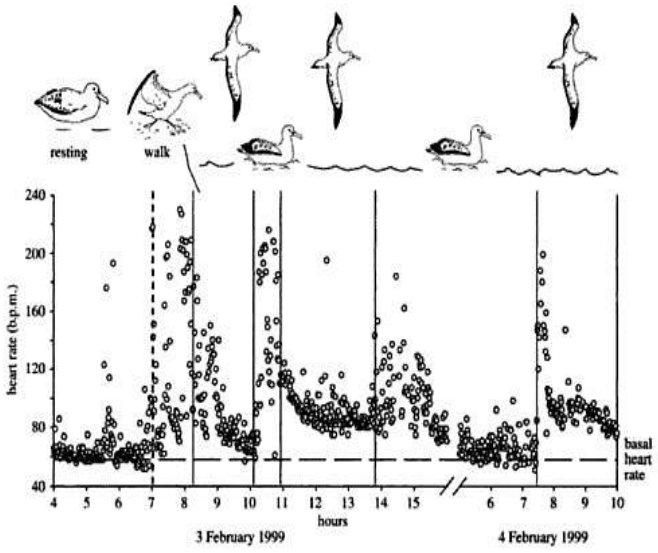

Figure 18: Heart Rate Levels of Albatross Bird [106]

In WSNs, the quick energy depletion of sensor nodes is mainly caused by complex routing processes and data transmission, resulting in early dead node occurrences and loss of network connectivity. To enhance the network lifetime, provide high energy saving, and achieve network stability in the presence of defective nodes in WSN, we

were motivated to adopt the albatross bird's flying technique in our proposed routing algorithm.

In [106], Bradshaw stated that albatross birds, when heading north, fly in approximately 60^o anticlockwise loops and change to clockwise loops when heading south. We adopted the flying scheme in our proposed dynamic energy soaring scheme (DESS) to select cluster heads optimally based on Bradshaw's claims. This soaring technique between high energy level nodes and low energy level nodes from the same energy level set is done by 60^o shifting [105]. Hence, the proposed ALBATROSS scheme consists mainly of two correlated algorithms. One is adopted from our previous Fault Tolerance Metaheuristic-Based Scheme (FTMBS) that we have already used to solve the CPP [107]. However, we applied the nondominated sorting genetic algorithm (NSGA-II) to the solutions provided by the GA in FTMBS. The DESS is used for the cluster head selection process. We claim that adopting this dynamic soaring technique saves network energy and hence improves the network lifetime as energy is an essential issue in wireless sensor networks. This technique is fundamentally based on the node's energy and the node's position, unlike the k-way spectral clustering technique which is based on the node's energy and random selection of cluster heads [100]. Some missing metrics in [68] as the network resiliency and load balancing between controllers are also considered in our scheme. Up to our knowledge, the Dynamic Energy Soaring Scheme for cluster head selection is presented for the first time in our work. Without loss of generality, we assume the followings:

- A non-failure controller located at the sink called Croot, root controller.
- Random distribution of sensor nodes

- In-band control path and data plane path
- Root controller knows the location of every node
- All controllers include in their forwarding tables the alternative paths in the presence of failure for their associated nodes
- Homogeneous SDN controllers.
- The shifting scheme is anticlockwise towards the high energy level set and clockwise towards the low energy level set.

The flowchart of the dynamic energy soaring scheme (DESS) is shown in Figure 19. Here, the energy of nodes is an essential factor for the DESS algorithm. The nodes are sorted based on their energy levels. Nodes with energy levels higher than a given threshold are added to a set called high set and denoted by setH{}. Based on this set, nodes are again placed into two sets. The first set is the low energy level set denoted by LES, consisting of nodes having energy level above the minimum energy level (E_{rmin}) to be a cluster head (which is 0.1 J in our case), and less than half the initial energy level. The second set is the high energy level set denoted by HES, consisting of the rest of nodes. Then the algorithm finds the means (X_{mean},Y_{mean}) for both HES and LES sets. Then, for each node in both sets, the corresponding image node denoted by (X_{image},Y_{image}) is found by shifting 60^o anticlockwise for HES and clockwise for LES. Equations (20), (21), (22), and (23) illustrate the coordinates of the image node after shifting for HES and LES, respectively [108].

$$X_{image}=\cos(60)*(X-X_{mean})-\sin(60)*(Y-Y_{mean})+X_{mean} \quad (20)$$

$$Y_{image}=\sin(60)*(X-X_{mean})+\cos(60)*(Y-Y_{mean})+Y_{mean} \quad (21)$$

$$X_{image}=\cos(-60)*(X-X_{mean})-\sin(-60)*(Y-Y_{mean})+X_{mean} \quad (22)$$

$$Y_{image}=\sin(-60)*(X-X_{mean})+\cos(-60)*(Y-Y_{mean})+Y_{mean} \quad (23)$$

After finding the mean for both sets, the distance value between each image node and the mean node is calculated and the minimum one is chosen. Then, the closest node to this image node is selected as the cluster head. If the node belongs to the HES set, the algorithm will start again from LES for the next cluster head selection and vice versa. By adopting this dynamic soaring among different nodes' energies eligible to be cluster heads, the overall network lifetime is improved. The proposed ALBATROSS scheme is presented in Figure 20. As shown, the ALBATROSS algorithm incorporates the FTMBS for solving the controller placement problem adopted from our previous work [107] and the DESS algorithm illustrated in Figure 19.

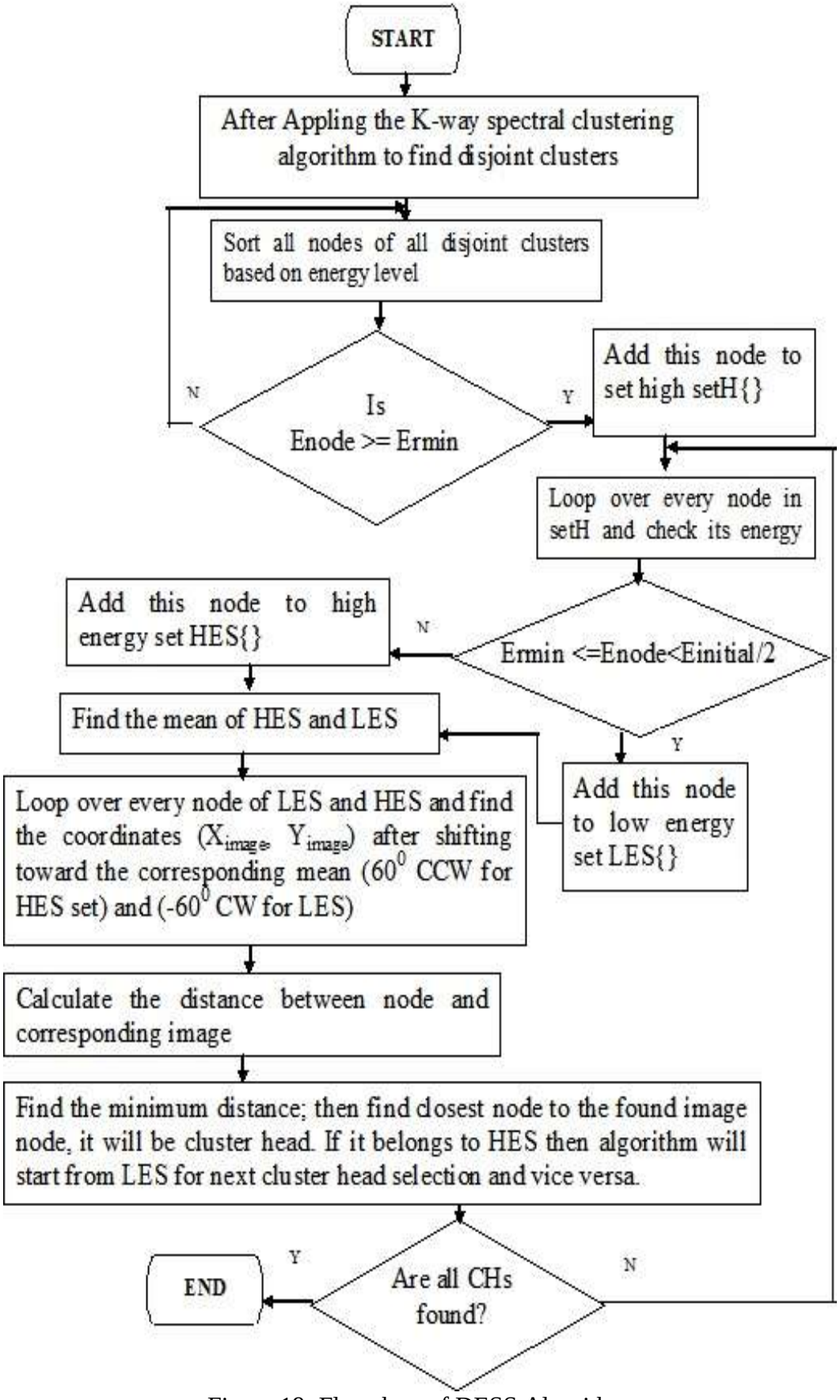

Figure 19: Flowchart of DESS Algorithm

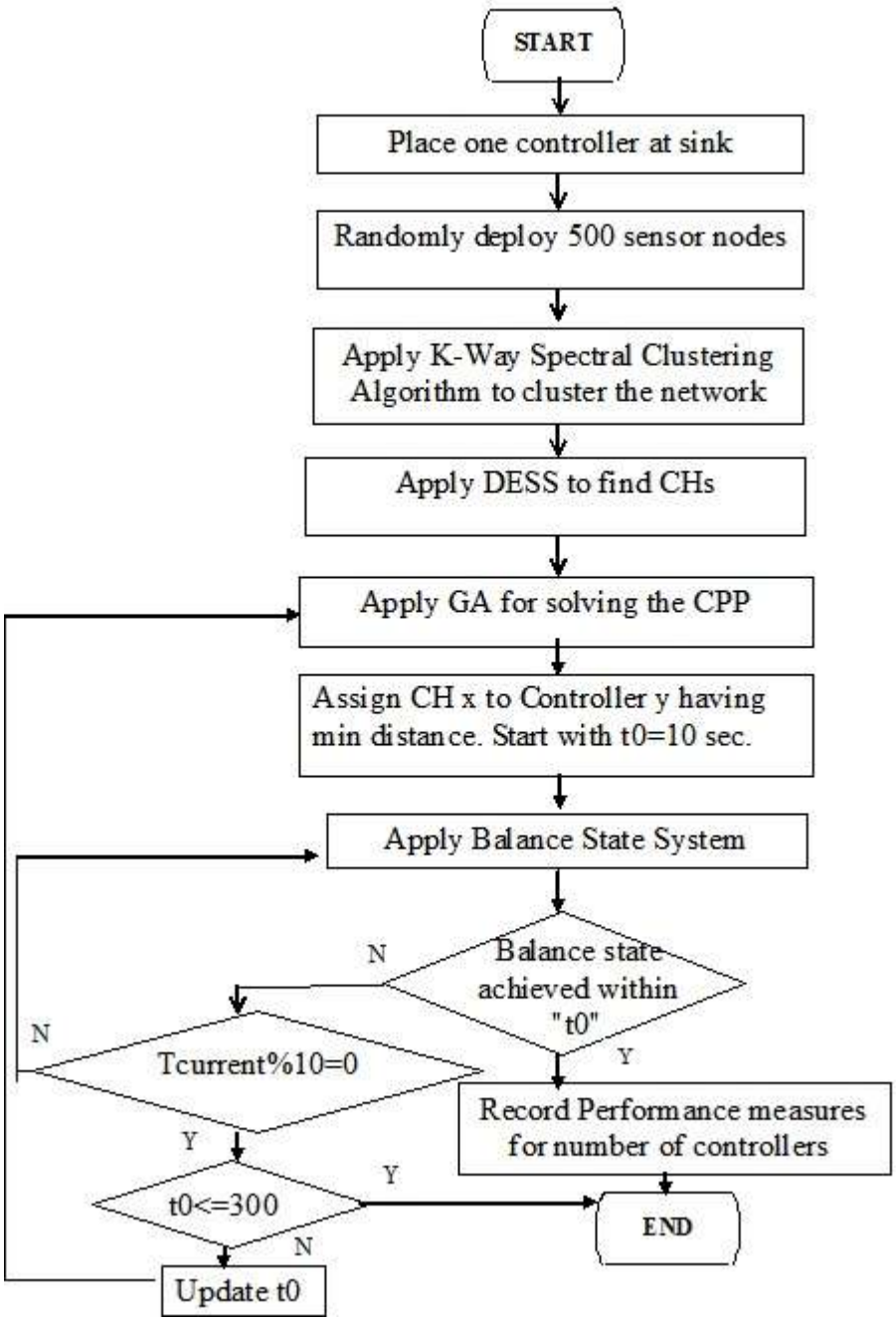

Figure 20: Flowchart of the ALBATROSS Scheme

5.2 Performance Evaluation

We have used MATLAB 2019b to run the proposed ALBATROSS scheme. We run the scheme 50 times for each fault percentage and took the average to achieve a 95% confidence interval. Three SDN controllers are used for 500 randomly deployed sensor nodes in a 200x200 m^2 field area under various faulty nodes' percentages.

Evaluation comparisons in terms of various network performance metrics were carried out against three energy-aware algorithms found in literature. The first algorithm proposed by the authors of [8] is a distributed cluster-based algorithm named as EEFCA, aiming to provide fault tolerance in the presence of CHs failures and whose cost function is based on nodes' energy and locations. Each sensor node does the cost function calculation to choose its cluster head among relay nodes having the highest cost value. The second algorithm is the GCEEC [95], an energy-based routing protocol. In GCEEC, the centroid position is considered for CHs election, and gateway nodes are selected from CHs to release the load from the overwhelming CHs and forward the data to the base station. The cluster head calculates the average cluster's energy and the weight for the gateway nodes in each cluster adjacent to the neighboring cluster head. The one with the highest weight is selected as a gateway node for the respective cluster. The third algorithm is FTMBS [107], a multi-objective algorithm aiming to achieve network steady-state, enhance load balance among controllers, increase throughput, and minimize the network delay in the presence of faulty nodes. Analysis of results carried is based on the following network parameter metrics:

5.2.1 Worst-Case Latency

Figure 21 presents the latency comparison of the algorithms mentioned above with the proposed ALBATROSS algorithm, where the worst-case latency is calculated using equation (6) in Chapter 3. As it is seen, the ALBATROSS scheme outperforms FTMBS, GCEEC, and EEFCA algorithms by 10%, 15%, and 20%, respectively. In EEFCA and GCEEC algorithms, calculations are done by sensor nodes and cluster heads, respectively, which add more delay to the network. In the FTMBS scheme, the root controller does all the required calculations. Specifically, it runs the GA to

select CH instead of the faulty node and optimally selects a controller instead of the faulty one. As a whole, this adds more delays, especially in the presence of more than one faulty CH. However, in the proposed ALBATROSS scheme, each controller runs the GA to select a CH instead of the faulty one. In this case, the controller does not wait for the main controller response, and hence, the overall network latency is decreased.

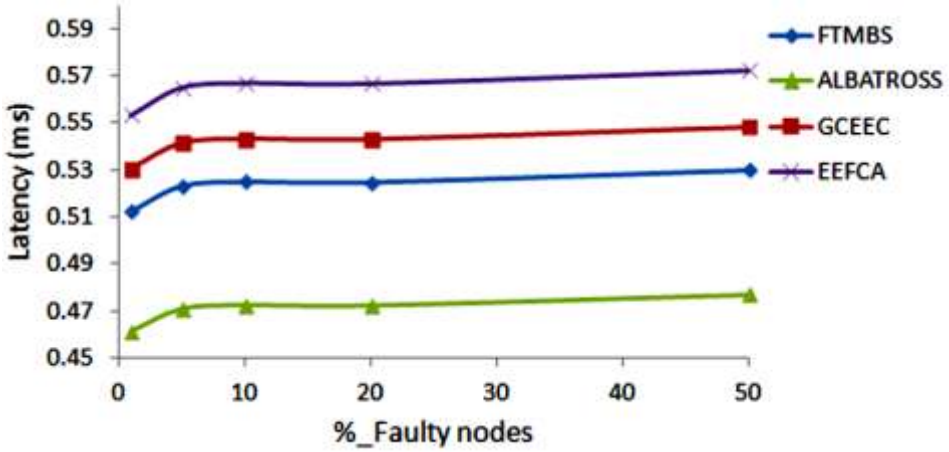

Figure 21: Latency Comparison

5.2.2 Network Lifetime

The network lifetime, given by equation (12) in Chapter 3, is a vital network performance metric, especially for wireless sensor networks. We have recorded the total number of alive nodes existing at the end of the simulation as a reflection of network lifetime under various faulty nodes' percentages. Balancing the energy consumption among the sensor nodes ensures elongated network functionality, and hence, increases the number of alive nodes at the end of the simulation. Figure 22 shows the total number of alive nodes for the proposed ALBATROSS scheme against FTMBS, GCEEC, and EEFCA algorithms. Energy depletion occurs faster in EEFCA and GCEEC algorithms due to excessive calculations done on behalf of

sensor nodes which causes early death of nodes. In FTMBS, the main controller selects the highest energy node to be a CH, and hence, the nodes' energy consumptions are not balanced. The nodes' energy consumption negatively affects the network stability. As it can be seen from Figure 22, the proposed ALBATROSS algorithm outperforms FTMBS, GCEEC, and EEFCA algorithms by 15%, 20%, and 25%, respectively.

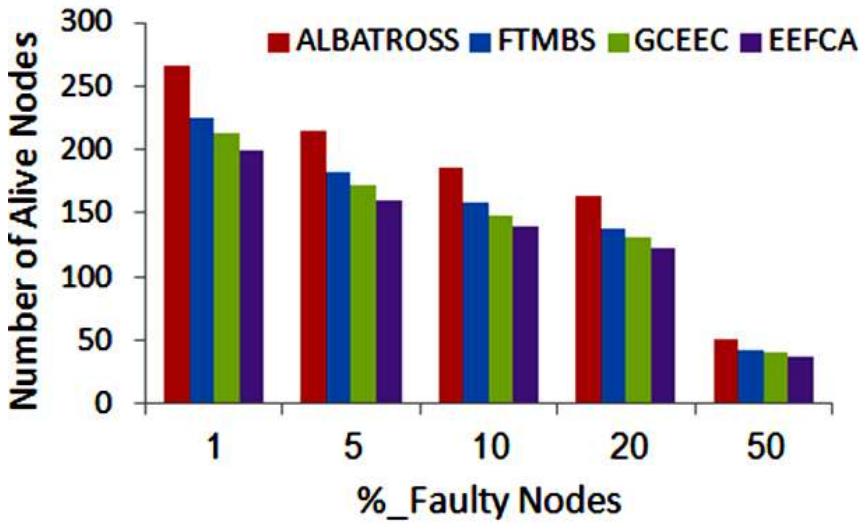

Figure 22: Network Lifetime Comparison

5.2.3 Percentage of Successfully Received Packets

One of the critical QoS factors is the percentage of successfully received packets given by equation (19) in Chapter 3. Figure 23 shows the percentage of successfully received packets under various faulty nodes' percentages. As expected, the percentage of successfully received packets decreases with the increase in faulty nodes. Since energy is balanced among the nodes, more alive nodes exist under the ALBATROSS scheme. Balancing the network energy has a positive impact on the percentage of successfully received packets. As seen from the figure, the

ALBATROSS scheme outperforms FTMBS, GCEEC, and EEFCA algorithms by 15%, 20%, and 25%, respectively.

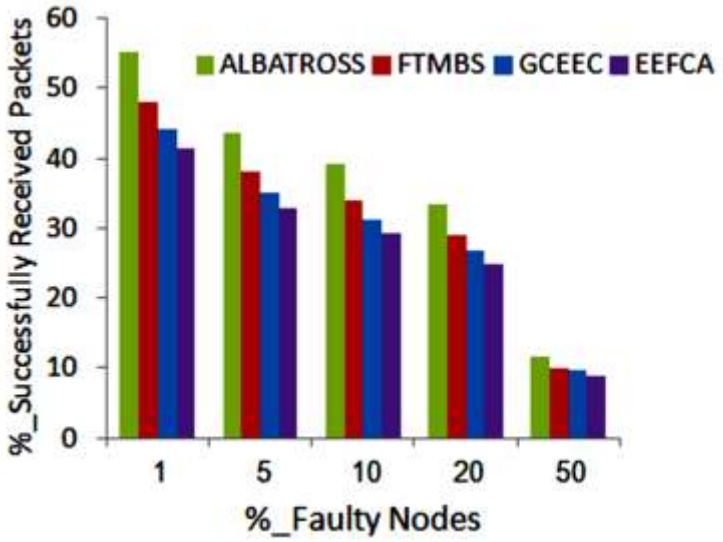

Figure 23: Percentage of Successfully Received Packets Comparison

5.2.4 Energy Consumption

Energy is a critical factor in wireless sensor networks since sensors are equipped with limited power in which data transmission consumes most of the sensors' energy. Although clustering helps to save the overall network energy, the efficient selection of cluster heads directly affects the network's energy consumption. ALBATROSS scheme is a cluster-based approach where the network is clustered into several disjoint clusters. In FTMBS [107], the cluster heads are selected from nodes having the highest energy level in each round. However, balancing the energy consumption among sensor nodes is not considered. In the ALBATROSS scheme, a cluster head is selected by applying the shifting technique described in the DESS algorithm which balances the sensor nodes' energy consumption. The soaring mechanism between the high-level energy nodes is done to balance the energy level among nodes. The energy consumption given by equation (9) in Chapter 3, is shown in Figure 24 under

various percentages of faulty nodes for the four algorithms. Note that when the percentage of faulty nodes increases, the number of dead nodes increases, and as a consequence, the total network energy consumption decreases.

On the other hand, due to sensor nodes' excessive calculation in both GCEEC and EEFCA algorithms, the network energy consumption is higher than that of the ALBATROSS algorithm. Results show that the ALBATROSS algorithm outperforms FTMBS, GCEECA, and EEFCA algorithms by 10%, 20%, and 25%, respectively. Hence, we can conclude that the ALBATROSS algorithm is an energy-efficient-based algorithm.

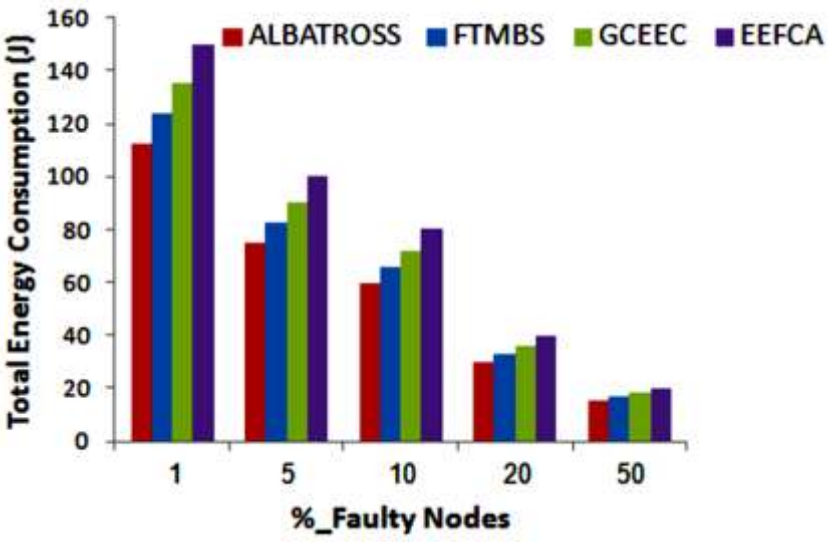

Figure 24: Network Energy Consumption Comparison

5.3 Applying ALBATROSS Scheme on Real Internet Topologies

We have applied the proposed ALBATROSS scheme on real datasets taken from the internet topology zoo [109]. We run the scheme to choose the best number of controllers starting at one controller to seven controllers for the JANET network and GEANT network. In an unweighted graph, the distance between two connected

nodes is the number of edges counted in the shortest path [26]. The network's diameter is defined as the maximum distance between any two connected nodes. We specified the latency constraint of a network as half of the diameter [26]. Hence, we set the latency constraints for Janet and GEANT networks as 14ms and 10m, respectively. Table 15 illustrates the latency and the execution time comparison results when different numbers of controllers denoted by N_CO are used. The proposed ALBATROSS scheme outperforms the two schemes [68] and [83] in terms of latency by 26% and 15%, respectively. It is worth to mention that since the ALBATROSS is a cluster-based network scheme, the overall latency decreases with the decrease in distance between nodes. Another fact is the ALBATROSS scheme dynamically chooses cluster heads by soaring among the nodes to prevent quick energy depletion. Hence, it increases the network lifetime. Conversely, as shown in the last main column of Table 15, the average execution time of the ALBATROSS scheme exceeds that of [68] and [83] by almost 5% and 7%, respectively.

Table 15: Latency and Execution Time Comparison

N_CO	Topology	Latency (ms)			Execution Time (s)		
		ALBATROSS	Ref. [68]	Ref. [83]	ALBATROSS	Ref. [68]	Ref. [83]
3	JANET	12	14	-	11	11	-
	GEANT	15.8	-	16.8	10	-	10
4	JANET	10	14.1		12	12	-
	GEANT	14	-	16.5	13	-	13
5	JANET	10.2	15	-	22	21	-
	GEANT	14.3	-	16.6	19	-	18
6	JANET	11	17		35	34	-
	GEANT	14.4	-	17	34	-	33
7	JANET	13	18	-	46	45	-
	GEANT	14.5	-	17.5	45	-	43
8	JANET	13.8	18.5		57	55	-
	GEANT	14.7	-	18	56	-	54
9	JANET	14	19	-	66	63	-
	GEANT	15.1	-	18.5	67	-	64

5.4 Computational Complexity

The computational complexity of the ALBATROSS is determined by the metaheuristic algorithm execution which is in $O(M \cdot popsize^2)$, where M is the number of sub-objectives and popsize is the population size. To express the overall computational complexity of the scheme, we also have considered the number of iterations iter_max and the number of cluster heads, K, that are included in the chromosome structure. The complexity of the individual evaluation process is bounded by computing the load of the cluster heads in $O(|K|)$, computing the delay of the graph in $O(|K|)$, computing the network lifetime, which is in $O(|E|^2 \cdot |V|)$. The complexity of the DESS algorithm is bounded by the number of cluster heads $O(|K^2|)$. Therefore, the computational complexity is bounded roughly in $O(\text{iter\_max} \cdot (M \cdot popsize^2 + popsize \cdot |E|^2 \cdot |V|. |K|^2) + |K|^2))$,which is acceptable for the proposed network structure.

5.5 Discussion

Heuristic algorithms are commonly used in research studies to solve NP-Hard problems in a reasonable amount of time. When search space is too complex, such as the GEANT network, which consists of 40 nodes, and requires 91390 combinations for placing four controllers. Therefore, evolutionary algorithms are the most suitable means to solve such problems. Most researchers have used metaheuristic algorithms for solving the CPP to optimize the network performance metrics. However, efficiently saving the network's energy with the presence of defective nodes was rarely considered. This motivates us to adopt the albatross bird's natural dynamic energy soaring scheme in the proposed DESS algorithm. The DESS effectively balances the network energy consumption by soaring among the high and low energy level nodes to select the network cluster heads. This soaring process for the cluster head selection involves shifting by 60^{o} counterclockwise for the high energy level nodes and clockwise for the low energy level nodes. The cluster head selection is essential since the controllers are chosen among these cluster heads by applying the GA algorithm. Actually, the presented ALBATROSS scheme is a modification to the FTMBS presented in Chapter 3. In FTMBS, the main controller keeps choosing the highest energy level node to be CH which may lead to an imbalance in energy consumption on one hand, and selection of new CH in presence of faulty CH for each cluster on the other hand. In the ALBATROSS scheme, the main controller only chooses the controllers instead of the faulty ones, and the CH selection is done on behalf of the associated controller. The proposed ALBATROSS scheme outperforms the FTMBS, GCEEC, and EEFCA schemes in terms of the network lifetime, percentage of successfully received packets, latency, and energy consumption. The ALBATROSS scheme also showed latency improvement over [68] and [83] when

applied over Janet and GEANT networks, respectively. The obtained network performance improvements added marginal increase in the execution time because of involving more calculations in the DESS algorithm.

Chapter 6

CONCLUSION

The presence of faulty nodes in wireless sensor networks is an expected natural event. Such event directly affects the network performance which has been rarely addressed in most researches. Maintaining the network's steady-state positively affects the network performance which can mainly be achieved by balancing the load among controllers. The proposed FTMBS scheme is the first to address various network performance metrics under different faulty nodes percentages. The FTMBS has shown its feasibility by improving the network QoS and network performance parameters. For instance, the network energy and reliability were improved with better connectivity and throughput. The BSS algorithm, a fundamental component in FTMBS, positively affects different network performance metrics under the GA and GRASP algorithms. However, the network performance improvement under GA and GRASP algorithms varies when three and five controllers are present. When three controllers are used, the BSS noticeably improves the average load of controllers under the GA by 10%, whereas this improvement is 6% under the GRASP algorithm. However, when five controllers are used, these improvements are found as 16% and 13% respectively. Also, when three controllers are present, the BSS improves the network lifetime under GA by 13%, whereas this improvement is 11% under the GRASP algorithm. when three controllers are present. However, when five controllers are used, these improvements are found as 24% and 20% respectively. On the other hand, the BSS improvements of the latency and percentage of successfully

received packets are slightly better under GRASP algorithm for both three and five controllers cases. Taking into consideration the obtaind average percentage of improvments and the predefined criteria for the selection of five controllers, we recommend the use of three controllers. The efficiency of the proposed FTMBS is verified using NSGA-II which shows the approximation of GA solutions to the Pareto frontier.

It is well known that network lifetime is a critical factor in WSNs which is directly affected by the used clustering technique. Based on this fact, we have proposed the ALBATROSS scheme, which mimics the dynamic soaring technique of the albatross bird to choose the network cluster heads by soaring among the high energy level nodes and low energy level nodes for solving the CPP. Actually, the ALBATROSS scheme is a modification to the FTMBS scheme. However, in FTMBS, the main controller keeps choosing the highest energy level node to be CH, which leads to an imbalance in energy consumption, and new CHs instead of the faulty CHs. In the proposed ALBATROSS scheme, the main controller only chooses the controllers instead of the faulty ones, and the CH selection is done on behalf of the associated controller. The results show that the ALBATROSS scheme is an energy-efficient scheme that outperforms the FTMBS, GCEEC [95], and EEFCA [8] algorithms by 10%, 20%, and 25%, respectively.

Besides the FTMBS and ALBATROSS schemes, we have also presented the ERQTM scheme (described in Appendix A) as a practical example of using the SDN concept in wireless body area networks. The ERQTM scheme incorporates two algorithms. The first is an energy-efficient routing algorithm which balances the energy among the sensor nodes and optimally selects the network's cluster heads.

The second is a QoS-supported traffic management algorithm which prioritizes the emergency data transmission to guarantee network reliability. Simulations results showed that the proposed ERQTM algorithm achieved QoS by maintaining high throughput, minimum delay, and maximum network lifetime compared to existing energy-efficient algorithms found in literature.

As future work, we plan to include the energy harvesting concept in our proposed schemes, where the sensor nodes are designed to harvest energy from the environment during the daytime and only utilize their battery during the night.

REFERENCES

[1] Y. Duan, Y. Luo, W. Li, P. Pace and G. Fortino, "Software Defined Wireless Sensor Networks: A Review," in *Proceedings of the 2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design*, Nanjing, 2018.

[2] D. Kreutz, F. Ramos, P. Verissimo, C. Rothenberg, S. Azodolmolky and S. Uhlig, "Software-Defined Networking:A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, 2015.

[3] H. Kim and N. Feamster, "Improving Network Management with Software Defined Networking," *IEEE Commun. Mag.*, vol. 51, no. 2, p. 114–119, 2013.

[4] I. Haque and N. Abu-Ghazaleh, "Wireless Software Defined Networking: A Survey and Taxonomy," vol. 18, no. 4, pp. 2713 - 2737, 19 May 2016.

[5] S. Costanzo, L. Galluccio, G. Morabito and S. Palazzo, "Software Defined Wireless Networks: Unbridling SDNs.," in *Proceedings of the 2012 European Workshop on Software Defined Networking*, Germany, 2012.

[6] H. Dhawan and S. Waraich, "A Comparative Study on LEACH Routing Protocol and its Variants in Wireless Sensor Networks: A Survey," *International Journal of Computer Applications*, vol. 95, no. 8, June 2014.

[7] M. Abo-Zahhad, O. Amin, M. Farrag and A. Ali, "Survey on Energy Consumption Models in Wireless Sensor Networks," *Open Transactions on Wireless Communications,* vol. 1, no. 1, pp. 23-40, 2014.

[8] K. Nitesh, M. Azharuddin and P. Jana, "Energy Efficient Fault-Tolerant Clustering Algorithm for Wireless Sensor Networks," in *International Conference on Green Computing and Internet of Things (ICGCIoT)*, Greater Noida, India, 2015.

[9] T. Luo, H. P. Tan and T. Q. S. Quek, "Sensor OpenFlow: Enabling Software-Defined Wireless Sensor Networks," *IEEE Communications Letters,* vol. 16, no. 11, pp. 1896-1899, 2012.

[10] H. I. Kobo, A. M. Abu-Mahfouz and G. P. Hancke, "A survey on software-defined wireless sensor networks: Challenges and design requirements," *IEEE Access,* vol. 5, p. 1872–1899, 2017.

[11] A. El-Mougy, M. Ibnkahla and L. Hegazy, "Software-Defined Wireless Network Architecture for the Internet-of-Things," in *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)*, Clearwater Beach, Florida, 2015.

[12] J. A. Puente Fernandez, L. J. Garcia Villalba and T.-H. Kim, "Software Defined Networks in Wireless Sensor Architectures: A Review," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in*

Design ((CSCWD)), Nanjing, 2018.

[13] W. Ejaz, M. Naeem, M. Basharat, A. Anpalagan and K. Sithamparanathan, "Efficient Wireless Power Transfer in Software-Defined Wireless Sensor Networks," *IEEE Sensors Journal*, vol. 16, no. 20, p. 7409–7420., 2016.

[14] A. Gante, M. Aslan and A. Matrawy, "Smart wireless sensor network management based on software-defined networking.," in *2014 27th Biennial Symposium on Communications (QBSC)*, Kingston, ON, 2014.

[15] R. Mohapatra, S. Mishra and T. Mohapatra, "Coverage problem in wireless sensor networks," *Comparative Cytogenetics*, vol. 2, no. 1, p. 67–72, 2012.

[16] G. Arumugam and T. Ponnuchamy, "Ee-leach: development of energy efficient leach protocol for data gathering in wsn," *Eurasip Journal on Wireless Communications & Networking*, vol. 2015, no. 1, p. 1–9, 2015.

[17] C. M. S. Figueiredo, A. L. dos Santos, A. A. F. Loureiro and J. M. Nogueira, "Policy-based adaptive routing in autonomous wsns," in *IEEE Ambient Networks International Conference on Distributed Systems: Operations and Management*, Barcelona, Spain , 2005.

[18] S. Shanmugapriya and M. Shivakumar, "Context based route model for policy based routing in wsn using sdn approach," *iSRASE*, vol. 4, no. 1, pp. 1-8, 2015.

[19] C. Wang, K. Sohraby, B. Li, M. Daneshmand and Y. Hu, "A survey of transport protocols for wireless sensor networks," *IEEE Network,* vol. 20, no. 3, p. 34–40, 2006.

[20] D. Tian and N. Georganas, "A node scheduling scheme for energy conservation in large wireless sensor networks," *Wireless Communications and Mobile Computing,* vol. 3, no. 2, p. 271–290, 2003.

[21] C. Hua and T. P. Yum, "Asynchronous random sleeping for sensor networks," *ACM Transactions on Sensor Networks,* vol. 3, no. 3, pp. 1-25, 2007.

[22] J. Levendovszky, K. Tornai, G. Treplan and A. Olah, "Novel load balancing algorithms ensuring uniform packet loss probabilities for wsn," in *Vehicular Technology Conference (VTC),* Budapest, Hungary, 2011.

[23] Y. Zhang, G. Sun and W. Li, "DEHCA: Load Balance Clustering Algorithm for Energy Heterogeneous WSN based on Distance," *Applied Mechanics & Materials,* Vols. 44-47, p. 3294–3298, 2010.

[24] C. Yoon, T. Park, S. Lee, H. Kang, S. Shin and Z. Zhang, "Enabling security functions with sdn: A feasibility study," *Computer Networks,* vol. 85, p. 19–35, 2015.

[25] W.-S. Kim and S.-H. Chung, "Proxy SDN Controller for Wireless Networks,"

Mobile Information Systems, vol. 2016, no. 4, pp. 1-14, 2016.

[26] G. Schutz, "A k-Cover Model for Reliability-Aware Controller Placement in Software-Defined Networks," in *Computational Science-ICCS 2019*, Springer International Publishing, 2019, pp. 604-613.

[27] Y. Hu, W. Wendong, X. Gong, X. Que and C. Shiduan, "Reliability-aware Controller Placement for Software-Defined Networks," in *IFIP/IEEE Symposium on Integrated Network Management*, Ghent, 2013.

[28] S. Milardo, L. Galluccio, G. Morabito and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *IEEE Conference on Computer Communications (INFOCOM)*, Kowloon, 2015.

[29] ON.Lab, "Introducing ONOS - a SDN network operating system for Service Providers [White Paper]," November 2014. [Online]. Available: https://docplayer.net/6967056-Introducing-onos-a-sdn-network-operating-system-for-service-providers.html.

[30] B. Oliveria, C. Margi and L. Gabriel, "TinySDN:Enabling Multiple Controllers for Software-Defined Wireless Sensor Networks," *IEEE Latin America Transactions*, vol. 13, no. 11, pp. 3690-3696, 2015.

[31] H. Mostafaei and M. Menth, "Software-defined wireless sensor networks: A

survey," *Journal of Network and Computer Applications,* vol. 119, pp. 42-56, 2018.

[32] R. D. Tubagus, "Performance Evaluation of a Software-Defined Wireless Sensor Network (Master of Science Thesis)," 2017. [Online]. Available: http://resolver.tudelft.nl/uuid:b04e6d40-05d1-47b6-b854-f19d3d5f520f.

[33] Y. Wang, H. Chen, X. Wu and L. Shu, "An energy-efficient SDN based sleep scheduling algorithm for WSNs," *Journal of Network and Computer Applications,* vol. 2016, no. 59, p. 39–45, 2015.

[34] D. Zeng, P. Li, S. Guo, T. Miyazaki, J. Hu and Y. Xiang, "Energy Minimization in Multi-Task Software-Defined Sensor Networks," *IEEE Transactions on Computers,* vol. 64, no. 11, pp. 3128-3139, 2015.

[35] W. Xiang, N. Wang and Y. Zhou, "An Energy-Efficient Routing Algorithm for Software-Defined Wireless Sensor Networks," *IEEE Sensors Journal,* vol. 20, no. 16, p. 7393–7400, 2016.

[36] H. Silva, A. Pereira, Y. Solano, B. Oliveira and C. Margi, "WARM: WSN application development and resource management," in *XXXIV Simposio Brasileiro de Telecomunicacoes* , Brazil, 2016.

[37] H. Fotouhi, M. Vahabi, A. Ray and M. Björkman, "An SDN-based traffic aware protocol for wireless sensor networks," in *IEEE 18th International*

Conference on e-Health Networking, Applications and Services (Healthcom)., Munich, Germany, 2016.

[38] Z. Qin, G. Denker, C. Giannelli, P. Bellavista and N. Venkatasubramanian, "Software-defined Wireless Sensor Networks and Internet of Things standardization synergism," in *IEEE Conference on Standards for Communications and Networking (CSCN).*, Tokyo, 2015.

[39] A. A. Lysko, L. Mamushiane and J. Mwangama, "Given a SDN Topology, How Many Controllers are Needed and Where Should They Go?," in *IEEE Conference on Network Function Virtualization & Software Defined Networks (IEEE NFV-SDN)*, Italy, 2018.

[40] G. Wang, y. Zhao, J. Huang and W. Wang, "The Controller Placement Problem in Software Defined Networking: A Survey," *IEEE Network*, vol. 31, no. 5, pp. 21-27, 2017.

[41] B. Heller, R. Sherwood and N. McKeown, "The Controller Placement Problem," in *Proceedings of the first workshop on Hot topics in software defined networks*, Finland, 2012.

[42] A. Upadhyay, "https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/," Info GIS MAP. [Online]. [Accessed 2020].

[43] J. A. Bondy and U. Murty, Graph Theory with Applications, Britain: The

Macmillan Press Ltd., 1976.

[44] J. Renfree, "Distance calculation using Haversine formula," [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/27785-distance-calculation-using-haversine-formula. [Accessed 2020].

[45] F. Hu, Q. Hao and K. Bao, "A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation," *IEEE Communication Surveys and Tutorials,* vol. 16, no. 4, pp. 2181-2206, 2014.

[46] O. Flauzac, C. Gonzalez and F. Nolot, "Developing a Distributed Software Defined Networking Testbed for IoT," *Procedia Computer Science,* vol. 83, pp. 680-684, 2016.

[47] B. Othmane, M. Ben Mamoun and R. Benaini, "An Overview on SDN Architecture with Multiple Controllers," *Journal of Computer Networks and Communications,* vol. 2016, no. 2, pp. 1-8, 2016.

[48] J. Cui, Q. Lu, H. Zhong, M. Tian and L. Liu, "A Load-Balancing Mechanism for Distributed SDN Control Plane using Response Time," *IEEE Transactions on Network and Service Management,* vol. 15, no. 4, pp. 1197 - 1206, 2018.

[49] S. Champagne, T. Makanju and C. Yao, "A Genetic Algorithm for Dynamic Controller Placement in Software Defined Networking," in *GECCO'18 Companion: Proceeding of the Genetic and Evolutionary Computation*

Conference Companion, Japan, 2018.

[50] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner and P. Tran-Gia, "Pareto-Optimal Resilient Controller Placement in SDN-based Core Networks," in *25th International Teletraffic Congress (ITC)*, Shanghai, 2013.

[51] S. Lange, S. Gebert, T. Zinner and P. Tran-Gia, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4-17, 2015.

[52] A. Jalili, M. Keshtgari, V. Ahmadi and M. Kazemi, "Controller Placement in Software Defined WAN Using Multi Objective Genetic Algorithm," in *2015 2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, Tehran, 2015.

[53] Cisco, "The Cisco Application Policy Infrastructure Controller," 2014. [Online]. Available: https://www.cisco.com/c/en/us/products/cloud-systems-management/application-policy-infrastructure-controller-apic/index.html#~for-partners. [Accessed 2020].

[54] A. Jalili, M. Keshtgari and R. Akbari, "A new Set Covering Controller Placement Problem Model for Large Scale SDNs," *Journal of Information Systems and Telecommunication*, vol. 6, pp. 25-32, 2018.

[55] T. FEO and M. RESENDE, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization,* vol. 6, pp. 109-133, 1995.

[56] J. E. C. Arroyo, V. d. O. Matos, A. G. d. Santos and L. B. Goncalves, "A GRASP Based Algorithm for Efficient Cluster Formation in Wireless Sensor Networks," in *IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Barcelona, 2012.

[57] S.-K. Yoon, Z. Khalib, N. Yaakob and A. Amir, "Controller Placement Algorithms in Software Defined Network- A Review of Trends and Challenges," in *MATEC Web of Conferences , ICEESI*, Malaysia, 2017.

[58] T. Hu, Z. Guo, T. Baker and J. Lan, "Multi-controller Based Software-Defined Networking: A Survey," *IEEE ACCESS*, vol. 6, pp. 15980 - 15996, 2018.

[59] D. Sembroiz, B. Ojaghi, D. Careglio and S. Ricciardi, "A GRASP Meta-Heuristic for Evaluating the Latency and Lifetime Impact of Critical Nodes in Large Wireless Sensor Networks," vol. 9, no. 21, p. 4564, 2019.

[60] B. Xiong, X. Peng and J. Zhao, "A Concise Queuing Model for Controller Performance in Software-Defined Networks," *Journal of Computers,* vol. 11, no. 3, pp. 232-237, 2016.

[61] Y. Zhang, N. Beheshti and M. Tatipamula, "On Resilience of Split-Architecture Networks," in *IEEE Global Telecommunications Conference* -

GLOBECOM 2011, Houston, TX, 2011.

[62] A. Sallahi and M. Hilaire, "Optimal Model for the Controller Placement Problem in Software Defined Networks," *IEEE Communications Letters,* vol. 19, no. 1, pp. 30-33, 2015.

[63] Y. Liu, A. Liu, Y. Hu, Z. Li, Y.-J. Choi, H. Sekiya and A. J. Li, "FFSC: An Energy Efficiency Communications Approach for Delay Minimizing in Internet of Things," *Green Communications and Networking For 5G Wireless,* vol. 4, pp. 3775-3793, 2016.

[64] R. Huang, X. Chux, J. Zhang and Y. Hu, "Energy-efficient monitoring in software defined wireless sensor networks using reinforcement learning: A prototype," *International Journal of Distributed Sensor Networks,* vol. 5, 2015.

[65] H. Yao, C. Qiu, C. Zhao and L. Shi, "A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks," *International Journal of Distributed Sensor Networks,* vol. 2, pp. 1-8, 2015.

[66] Wikipedia, "Multi-objective optimization," 19 February 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Multi-objective_optimization&oldid=941634844.

[67] N. Gunantara, "A Review of Multi-Objective Optimization:Methods and its

Applications," *Cogent Engineering*, vol. 5, no. 1, 2018.

[68] S. Mohanty, P. Priyadarshini, S. Sahoo, B. Sahoo and S. Sethi, "Metaheuristic Techniques for Controller Placement in Software-Defined Networks," in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, Kochi, India, 2019.

[69] M. Ouzzif, J.-M. Sanner, Y. Hadjadj-Aoul and G. Rubino, "An Evolutionary Controllers' Placement Algorithm for Reliable SDN Networks," in *13th International Conference on Network and Service Management (CNSM)*, Tokyo, Japan, 2017.

[70] S. Lin, "NGPM - A NSGA-II Program in Matlab v1.4," MATLAB Central File Exchange, 16 July 2011. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/31166-ngpm-a-nsga-ii-program-in-matlab-v1-4.

[71] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182-197, 2002.

[72] A. Herrera-Poyatos and F. Herrera, "Genetic and Memetic Algorithm with Diversity Equilibrium based on Greedy Diversification," arXiv:1702.03594 [cs.AI], 2017.

[73] B. B. Lokesh and N. Nalini, "Genetic Algorithm Based Node Fault Detection and Recovery in Distributed Sensor Networks," *I.J.Computer Network and Information Security,* vol. 12, pp. 37-46, 2014.

[74] A. Ateya, A. Muthanna, A. Vybornova, A. Algarni, A. Abuarqoub, Y. Koucheryavy and A. Koucheryavy, "Chaotic salp swarm algorithm for SDN multi-controller networks," *Engineering Science and Technology, an International Journal,* vol. 22, pp. 1001-1012, 2019.

[75] S.-B. Kang and G.-I. Kwon, "Load Balancing Strategy of SDN Controller Based on Genetic Algorithm," *Advanced Science and Technology Letters ,* vol. 129, pp. 219-222, 2016.

[76] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li and T. Li, "Density Cluster Based Approach for Controller Placement Problem in Large-Scale Software Defined Networkings," *Computer Networks,* vol. 112, pp. 24-35, 2017.

[77] H. Selvi, G. Gur and F. Alagoz, "Cooperative Load Balancing for Hierarchical SDN Controllers," in *IEEE 17th International Conference on High Performance Switching and Routing,* Yokohama, 2016.

[78] G. Yao, J. Bi, Y. Li and L. Guo, "On the Capacitated Controller Placement Problem in Software Defined Networks," vol. 18, no. 8, 2014.

[79] Y. Hu, W. Wang, X. Gong and X. Que, "BalanceFlow: Controller load

balancing for OpenFlow networks," in *IEEE 2nd International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Hangzhou, 2012.

[80] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman and R. R. Kompella, "ElastiCon: An Elastic Distributed SDN Controller," in *IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, Marina del Rey, CA, 2014.

[81] A. Ruiz-Rivera, K.-W. Chin and S. Soh, "GreCo:An Energy Aware Controller Association Algorithm for Software Defined Networks," *IEEE Communications Letters*, vol. 19, no. 4, pp. 541-544, 2015.

[82] L. Wang and X. Yang, "SDN Load Balancing Method Based on K-Dijkstra," *International Journal of Performability Engineering*, vol. 14, no. 4, pp. 709-716, 2018.

[83] B. Killi and S. Rao, "Capacitated Next Controller Placement in Software Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 514 - 527, 2017.

[84] J. Yu, Y. Wang, K. Pei, S. Zhang and J. Li, "A Load Balancing Mechanism for Multiple SDN Controllers based on Load Informing Strategy," in *18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Japan, 2016.

[85] Y. Hu, T. Luo, W. Wang and C. Deng, "On the Load Balanced Controller Placement Problem in Software Defined Networks," in *International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2016.

[86] T. Hu, J. Lan, J. Zhang and W. Zhao, "EASM: Efficiency-Aware Switch Migration for Balancing Controller Loads in Software-Defined Networking," *Peer-to-Peer Networking and Applications*, vol. 12, pp. 452-464, 2018.

[87] Y. Hu, T. Luo, N. C. Beaulieu and C. Deng, "The Energy-Aware Controller Placement Problem in Software Defined Networks," *IEEE Communications Letters*, vol. 21, no. 4, pp. 741 - 744, April 2017.

[88] A. Abdelaziz, A. Fong, A. Gani, U. Garba, S. Khan, A. Akhunzada, H. Talebian and K.-K. Raymond Choo, "Distributed Controller Clustering in Software Defined Networks," *PLOS*, vol. 12, no. 4, 6 April 2017.

[89] G. Wang, Y. Zhao, J. Huang and Y. Wu, "An Effective Approach to Controller Placement in Software Defined Wide Area Networks," *IEEE Transactions on Network and Management*, vol. 15, no. 1, 2018.

[90] G. Wang, Y. Zhao, J. Huang and R. M. Winter, "On the data aggregation point placement in smart meter networks," in *26th International Conference Computer Communication Network (ICCCN)*, Vancouver, BC, Canada, 2017.

[91] E. Dijkstra, "A note on two problems in connexion with graphs.," *Numerische Mathematik* , vol. 1, p. 269–271, 1959.

[92] H. Yannan, W. Wendong, G. Xiangyang, Q. Xirong and C. Shiduan, "On the Placement of Controllers in Software-Defined Networks," *The Journal of China Universities of Posts and Telecommunications,* vol. 19, no. 2, pp. 92-97, 2012.

[93] WIKIPEDIA, "Brute-force search," 19 December 2020. [Online]. Available: https://en.wikipedia.org/wiki/Brute-force_search.

[94] Y. Hu, W. Wang, X. Gong, X. Que and S. Cheng, "On Reliability-Optimized Controller Placement for Software-Defined Networks," *China Communications,* vol. 11, no. 2, pp. 38-54, 2014.

[95] K. Qureshi, M. U. Bashir, J. Lloretq and A. Leon, "Optimized Cluster-Based Dynamic Energy-Aware Routing Protocol for Wireless Sensor Networks in Agriculture Precision," *Journal of Sensors,* p. 19 pages, 2020.

[96] A. K. Singh and S. Srivastava, "A Survey and Classification of Controller Placement Problem in SDN," *International Journal of Network Management,* vol. 28, no. 3, 2018.

[97] M. Resende and C. Ribeiro, "International Series in Operations Research & Management Science," in *Handbook of Metaheuristics,* vol. 57, Springer,

2003, pp. 219-249.

[98] V. Gungor, "A Forecasting-Based Monitoring and Tomography Framework for Wireless Sensor Networks," in *IEEE International Conference on Communications*, Istanbul, Turkey, 2006.

[99] J. Brownlee, "GitHub," [Online]. Available: http://www.cleveralgorithms.com/nature-inspired/stochastic/grasp.html. [Accessed 2015].

[100] A. Jorio, S. El Fkihi, B. Elbhiri and D. Aboutajdine, "A New Clustering Algorithm in WSN Based on Spectral Clustering and Residual Energy," in *Seventh International Conference on Sensor Technologies and Applications, SENSORCOMM 2013*, 2013.

[101] A. Aoullay, "Spectral Clustering for Beginners," 2018. [Online]. Available: https://towardsdatascience.com/spectral-clustering-for-beginners-d08b7d25b4d8.

[102] A. Y. Ng, M. I. Jordan and Y. Weiss, "On Spectral Clustering:Analysis and an algorithm," in *Advances in Neural Information Processing Systems*, 2002.

[103] U. Luxburg, "A Tutorial on Spectral Clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395-416, 2007.

[104] J. W. Emerson, W. A. Green, B. Schloerke, J. Crowley, D. Cook, H. Hofmann and H. Wickham, "The generalized pairs plot," *Journal of Computational and Graphical Statistics,* vol. 22, no. 1, pp. 79-91, 2013.

[105] J. Chu, "Engineers identify key to albatross' marathon flight," Massachusetts Institute of Technology MIT News Office, 10 October 2017. [Online]. Available: http://news.mit.edu/2017/engineers-identify-key-albatross-marathon-flight-1011.

[106] D. Bradshaw, in *Vertebrate Ecophysiology: An Introduction to its Principles and Applications*, Cambridge University Press, 2003.

[107] N. Samarji and M. Salamah, "A Fault Tolerance Metaheuristic-Based Scheme for Controller Placement Problem in Wireless Sensor Networks," *International Journal of Communication Systems,* vol. 34, no. 4, 2021.

[108] J. Vince, "Matrix Transforms," in *Foundation Mathematics for Computer Science*, Springer, 2015, pp. 193-194.

[109] M. Roughan and W. Willinger, "Internet Topology Research Redux," 2013. [Online]. Available: http://sigcomm.org/education/ebook/SIGCOMMeBook2013v1_chapter1.pdf.

[110] S. Movassaghi, M. Abolhasan, M. Lipman, D. Smith and A. Jamalipour, "Wireless Body Area Networks: A Survey," *IEEE Communications Surveys &*

Tutorials, vol. 16, no. 3, pp. 1658-1686, 2014.

[111] C. Otto, E. Jovanov and A. Milenkovic, "A WBAN-based system for health monitoring at home," in *3rd IEEE/EMBS International Summer School and Symposium on Medical Devices and Biosensors*, 2006.

[112] K. Y. Jamil and Y. R. Mehmet, "Wireless body area network (WBAN) for medical applications New Developments," in *New Developments in Biomedical Engineering,* vol. 1, IntechOpen, 2010, pp. 591-628.

[113] B. Zhen, H.-b. Li and R. Kohno, "Networking issues in medical implant communications," *International Journal of Multimedia and Ubiquitous Engineering,* vol. 4, no. 1, 2009.

[114] D. J. Vergados, D. D. Vergados and I. Maglogiannis, "Applying Wireless Diffserv for QoS Provisioning in Mobile Emergency Telemedicine," in *IEEE Globecom*, San Francisco, CA, USA, 2006.

[115] A. Sandhu and A. Malik, "PAP: Priority Aware Protocol for Healthcare Applications in Wireless Body Area Network," *International Journal of Recent Technology and Engineering (IJRTE),* vol. 8, no. 5, pp. 2733-2739, 2020.

[116] R. Cavallari, F. Martelli, R. Rosini, C. Buratti and R. Verdone, "A survey on wireless body area networks: Technologies and design challenges," *IEEE*

Communications Surveys and Tutorials, vol. 16, no. 3, p. 1635–1657, 2014.

[117] K. Hasan, A. Khandakar, K. Biswas and M. Saiful Islam, "Software-defined application-specific traffic management for wireless body area networks," *Future Generation Computer Systems*, pp. 274-285, 2020.

[118] M. Quwaider and S. Biswas, "DTN routing in body sensor networks with dynamic postural partitioning," *Ad Hoc Networks,* vol. 8, no. 8, p. 824–841, 2010.

[119] M. Quwaider and S. Biswas, "On-body packet routing algorithms for body sensor networks," in *First International Conference on Networks and Communications. NETCOM 09*, Chennai, India, 2009.

[120] C. Mohanty and S. Swayamsiddha, "Application of cognitive Internet of Medical Things for COVID-19 pandemic," *Diabetes & Metabolic Syndrome: Clinical Research & Reviews,* vol. 14, pp. 911-915, 2020.

[121] Y. Zhang, B. Zhang and S. Zhang, "A Lifetime Maximization Relay Selection Scheme in Wireless Body Area Networks," *Sensors,* vol. 17, no. 1267, 2017.

[122] *IEEE Standard for Local and Metropolitan Area Networks Part 15.6: Wireless Body Area Networks,* 2012.

[123] M. El Azhari, N. El Moussaid, A. Toumanari and R. Latif, "Equalized energy

consumption in wireless body area networks for a prolonged network lifetime," *Wireless Commun. Mobile Comput.*, 2017.

[124] S. Yahiaoui, M. Omar, A. Bouabdallah and E. Natalizio, "An energy efficient and QoS aware routing protocol for wireless sensor and actuator networks," *AEU-International Journal of Electronics and Communications,* vol. 83, 2018.

[125] E. E. Tsiropoulou, G. Mitsis and S. Papavassiliou, "Interest-aware energy collection & resource management in machine to machine communications," *Ad Hoc Networks,* vol. 68, pp. 48-57, 2018.

[126] D. Fabio, T. Ilenia and G. Yu, "1 Hop or 2 Hops: Topology Analysis in Body Area Network," in *Proceedings of the 2014 European Conference on Networks and Communications (EuCNC)*, Italy, 2014.

[127] S. R. Chavva and R. S. Sangam, "An energy-efficient multi-hop routing protocol for health monitoring in wireless body area networks," *Network Modelling Analysis in Health Informatics and Bioinformatics,* vol. 8:21, 2019.

[128] Y. Qu, G. Zheng, H. Wu, B. Ji and H. Ma, "An Energy-Efficient Routing Protocol for Reliable Data Transmission in Wireless Body Area Networks," *Sensors,* vol. 19:4238, 2019.

[129] W. R. Heinzelman, A. Chandrakasan and H. Balakris, "Energy-efficient communication protocol for wireless microsensor networks," in *In Proceeding*

of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI, USA, 2000.

[130] R. Goyal, R. B. Patel, H. Bhaduria and D. Prasad, "An Energy Efficient QoS Supported Optimization Transmission Rate Technique in WBANs," *Wireless Personal Communication*, vol. 117, p. 235–260, 2021.

[131] N. Javaid, M. Farid, Z. Khan, N. Alrajeh and Z. Abbas, "M-ATTEMPT: A new energy-efficient routing protocol for wireless body area sensor networks," *Procedia Computer Science*, vol. 19, pp. 224-231, 2013.

[132] A. Gulnaz, Z. Jianhua and M. M. S. Fareed, "PERA: Priority-based Energy-efficient Routing Algorithm for WBANs," *Wireless Personal Communication*, vol. 96, no. 3, 2017.

[133] Z. ULLAH, I. AHMED, F. A. KHAN, M. ASIF, M. NAWAZ, T. ALI, M. KHALID and F. NIAZ, "Energy-Efficient Harvested-Aware Clustering and Cooperative Routing Protocol for WBAN (E-HARP)," *IEEE ACCESS*, vol. 7, pp. 100036-100050, 2019.

[134] M. El Haziti, A. Bahae and J. Abdelillah, "Wireless body area networks: a comprehensive survey," *Journal of Medical Engineering & Technology*, vol. 44, no. 3, pp. 97-107, 2020.

[135] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*,

vol. 17, no. 4, pp. 395-416, 2007.

[136] H. Teng, A. Liu, X. Liu and H. Shen, "Adaptive Transmission Power Control for Reliable Data Forwarding in Sensor Based Networks," *Wireless Communications and Mobile Computing*, vol. 2018, no. 2, pp. 1-22, 2018.

[137] N. Samarji and M. Salamah, "ERQTM: Energy-efficient Routing and QoS-supported Traffic Management for SDWBANs," *IEEE Sensors Journal*, vol. 21, no. 14, 2021.

[138] D. Montgomery, "Simple Comparative Experiments," in *Design and Analysis of Experiments*, Arizona, John Wiley & Sons, 2001, pp. 45-51.

[139] S. Yousaf, S. Ahmed, M. Akbar, N. Javaid, Z. Khan and U. Qasim, "Co-CEStat: Cooperative Critical Data Transmission in Emergency in Static Wireless Body Area Network," in *2014 Ninth International Conference on Broadband and Wireless Computing, Communication and Applications*, Guangdong, China, 2014.

[140] C. Guo, R. Prasad and M. Jacobsson, "Packet Forwarding with Minimum Energy Consumption in Body Area Sensor Networks," in *7th IEEE Consumer Communications and Networking Conference*, Las Vegas, NV,USA, 2010.

[141] Y. Abdelmalek and T. Saadawi, "Destination-assisted routing enhancement (DARE) protocol for ad-hoc mobile networks," in *SARNOFF: Proceedings of*

the 32nd International Conference on Sarnoff Symposium, USA, 2009.

[142] O. Smail, A. Kerrar, Y. Zetili and B. Cousin, "ESR: Energy aware and stable routing protocol for WBAN networks," in *International Wireless Communications and Mobile Computing Conference (IWCMC)*, Paphos, Cyprus, 2016.

[143] Q. Nadeem, N. Javaid, S. Mohammad, M. Khan, S. Sarfraz and M. Gull, "SIMPLE: Stable Increased-Throughput Multi-hop Protocol for Link Efficiency in Wireless Body Area Networks," in *Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, Compiegne, France, 2013.

APPENDICES

Appendix A: Energy-efficient Routing and QoS-Supported Traffic Management Scheme for SDWBANs

1. Wireless Body Area Network (WBAN)

WBAN is integrated into various applications, including healthcare, military, personal entertainment, advanced sports training, and so forth. For instance, wireless body sensors measure the human body's physiological factors such as glucose, temperature, blood pressure, etc., and forward it to the concerned authorities using an intranet/internet facility [110]. This kind of continuous monitoring is essential, especially for patients with serious medical conditions who need medical intervention at any time. The wireless body sensors can be implanted either inside the human body or externally on the body surface. WBAN consists of light-weighted wireless body sensors and a coordinator, where the coordinator gathers the sensed data and transmits it to an external user or a remote server [111], [112]. Usually, these sensors are empowered with limited battery capacity difficult to be recharged or replaced in case of depletion. Consequently, the depleted senor will stop working. Additional burdens occur on human life and network lifetime. Hence, energy efficiency is one of the crucial parameters needed for a prolonged network lifetime. Another challenge in WBANs is achieving acceptable QoS levels. QoS includes high throughput requirements (1kb/s to 10Mb/s) and reliability [113], [114], [115], [116].

Reliability of transmission is a fundamental issue in WBANs revealed by guaranteed data delivery, especially for emergency data. Therefore, achieving reliability requires optimized throughput and data delivery with minimum end-to-end delay [114]. Accordingly, in life-critical scenarios where fast and reliable routes are mandatory for transmission [117], the reliability of emergency data transmission plays a crucial

role in saving a patient's life. Efficient routing is considered the backbone in achieving reliability [118], [119]. Hence, the most critical challenge faced by WBANs is achieving reliability and prolonged network lifetime. This challenge is nowadays a hot topic in cognitive radio-based internet of medical things applications (CIoMT) for coronavirus disease (COVID-19) patients. Due to this pandemic which caused a universal lockdown and movement restrictions, most health activities have gone online based on wireless communication and networking, which consume bandwidth [120]. CIoMT is based on dynamic spectrum allocation to accommodate the massive number of applications and devices. Hence, CIoMT is a panacea technology for rapid diagnosis, dynamic monitoring, tracking, better treatment, decision-making, and control without spreading the virus to others [120]. Thus, to achieve a high QoS, a well-designed routing algorithm and efficient network management are needed.

Clustering-based routing algorithms are appropriate for WBANs where network partitioning is implemented. Each partition or cluster contains one cluster head (CH), and the rest of the nodes are cluster members. Single-hop transmission occurs between CH and cluster members. CH gathers sensed data from its members and transmits it to the coordinator via single-hop or two-hop communication [121], [122]. The clustering approach ensures minimum energy utilization and offers maximum data delivery with appropriate end-to-end delay [123]- [124]. Tsiropoulou et al. [125] present an energy-efficient resource management approach by proposing a probability-based cluster formation incorporating the interest-based and distance-based factors in cluster formation. The proposed cluster-based scheme aims at

prolonging the network operation by allowing each device to harvest energy through radio frequency (RF) signals.

Maximizing the energy saving of sensor nodes is a critical challenge in WBAN. Several energy-aware routing schemes have been presented aiming at prolonging the network lifetime of wireless networks. Most of the power-saving approaches reduce the frequency of sending network control messages and redundant retransmissions. Some of these energy-aware routing schemes are listed in Table 16, which summarizes their pros and cons.

Table 16: Energy-Aware Routing Schemes

Approach	Pros	Cons
Co-CEStat [140]	High network throughput & network lifetime. Supports dynamic routing	Energy utilization is high
MEPF [141]	Minimize node's energy with less transmission power	Network latency is high
DARE [142]	Minimize node's energy consumption	Load is not uniformly distributed
ESR [143]	Supports patients mobility and handles traffic load changes	Network lifetime is low
SIMPLE [144]	Energy consumption is balanced. High network throughput	Packet loss is high
Proposed ERQTM	Energy consumption is balanced among all nodes. High network throughput, elongated network lifetime & minimal network latency. Supports WBAN QoS.	A bit high energy utilization due to high transmission power for emergency data

IEEE 802.15.6 [121] is designed explicitly for WBAN communication with two topologies: star topology or two-hop tree topology. The two-hop tree topology is based on a sensor node's cooperative transmission through a relay node to the sink or controller. This type of communication is done using Time Division Multiple Access

(TDMA) slots that help save the network energy and prolong the network lifetime [126]. Therefore, an appropriate relay selection is important in cooperative transmission strategy [121] to achieve high QoS.

In this part of the thesis, we have presented an ERQTM scheme for intra-WBANs to enhance the network lifetime and achieve network reliability and stability. Thus, we have addressed the problem as a joint scheme, developing an energy-efficient routing scheme (ER) and the QoS traffic management scheme (QTM). The ER scheme is a cluster-based energy-efficient routing algorithm aiming to elongate the network lifetime. The traffic management scheme ensures network reliability and stability by prioritizing the emergency data during transmission, thus ensuring high QoS. Most researchers prioritize emergency data by restricting low energy level nodes to send emergency data and avoid sending normal data. That causes degradation in the network throughput and failure in achieving high network QoS. In our work, we have overcome such shortcomings by proposing a meta-heuristic-based energy-efficient routing approach for cluster head selection. This approach avoids quick depletion of the network energy by dynamically selecting the network cluster heads. The dynamic cluster head selection causes a positive impact on the network lifetime and network QoS. Since clustering the nodes is an NP-hard problem, the proposed ER scheme is based on a genetic algorithm [73] to select the network cluster heads optimally. The cost model includes the following parameters: the nodes' residual energies, nodes' consumption energy rate, distance to the controller, signal to noise ratio, and path-loss effect. Since our scheme is based on intra-WBAN communication, body movements may cause path loss (i.e., data loss during transmission). Path-loss highly affects network throughput; therefore, it is included in the optimization process as

well. Besides the energy-efficient cluster-based routing algorithm, we have presented a priority-based scheme to manage the traffic flow. High priority is given to the emergency data (ED) for routing, and the associated nodes are assigned with a high transmission rate and less sensing interval. Since energy depletes quickly with high data rate transmission, our proposed scheme balances the energy among the sensor nodes. That is done by considering the nodes' energy consumption rates in consecutive intervals and their residual energies to optimally select the network cluster heads once the energy level falls under a certain threshold. Experimental results are conducted to evaluate the efficiency of the proposed ERQTM scheme.

The contributions of this work include the following:

• The proposed ERQTM scheme balances the energy among sensor nodes and achieves a high packet delivery rate. That is done by considering the nodes' energy consumption rates in consecutive intervals and their residual energies in the cluster head selection process. Hence, it avoids the quick depletion of the node's energy and early dead nodes' existence. That positively impacts the network lifetime on the one hand and the network reliability by ensuring the emergency data transmission on the other hand.

• Most routing algorithms in Literature focus on selecting the closest relay nodes to the sink for data transmission, which causes quick depletion of energy. Our proposed scheme optimally selects a CH using GA. The GA objective function focuses on nodes' residual energies, energy consumption rates, distance to the controller, and path-loss effect. Up to our knowledge, this is the first work in which the GA is used,

taking into consideration various QoS metrics to select an appropriate CH among the wireless body sensor nodes.

• Most of the literature studies do not focus on prioritizing the routing of critical or emergency data. We claim that instantaneous notification of any emergency data cuts off the delay to immediate medical treatment, saving the patient's life. Cutting off the routing delay is achieved by giving high priority to emergency data. In the traffic management part of our scheme, we have given high priority to emergency data with a high transmission rate so that it is immediately routed toward the controller.

• The obtained results demonstrate the efficiency of the proposed ERQTM scheme in achieving the WBAN's QoS. That is illustrated by achieving network stability and reliability through a high throughput rate, less delay, and elongated network lifetime.

2. System Overview

In this section, we present a cluster-based intra-SDWBAN communication system model that includes the following system assumptions:

- All sensors upon deployment have equal energy
- SDN controller node has unlimited energy
- All sensors either send normal data or emergency data
- Sensors transmit with different data rate according to the data type
- Intra-cluster: single hop between sensors and CH
- Inter-cluster: either single or two-hops communication between CHs and controller
- We assume events occurring during time interval follows a Poisson distribution

3. Network Model

Our system model incorporates an SDN controller implemented at the sink of WBAN consists of 15 sensor nodes deployed at the human body, as shown in Figure 25, which summarizes the topology of software-defined wireless body area network (SDWBAN). Our proposed energy-efficient routing and QoS supported traffic management (ERQTM) scheme consists of two correlated algorithms: energy-efficient routing algorithm (ER) and QoS traffic management algorithm (QTM). Before explaining these two algorithms, it is of great importance to determine the number of clusters needed in our proposed scheme. That is explained in section 3.1.

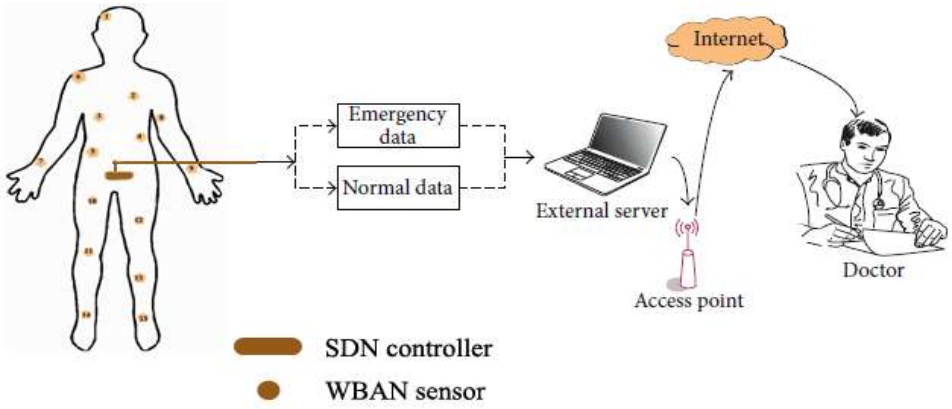

Figure 25: SDWBAN Topology

There are two types of WBAN transceivers: nRF 2401A and CC2420. We will use nRF 2401 in our simulation due to low power transmission and for comparison purposes with other techniques. Both transceivers have the same bandwidth, 2.4GHz. Table 17 shows details of these two kinds of WBAN sensors. Table 18 includes the sensor nodes' locations on the human body.

Table 17: Transceiver Energy Parameters **[127]**

Parameters	nRF 2401A	CC2420	Units
DC current (Tx)	10.5	17.4	mA
DC current (Rx)	18	19.7	mA
Supply voltage (min)	1.9	2.1	V
$E_{Tx\text{-}elec}$	16.7	96.9	nJ/bit
$E_{Rx\text{-}elec}$	36.1	172.8	nJ/bit
E_{amp}	$1.97e^{-9}$	$2.71e^{-7}$	j/b

Table 18: Nodes' Location

Node #	Description	Location	
		x-axis	y-axis
1	EEG	0.32	1.77
2	ECG	0.35	1.37
3	ECG	0.22	1.35
4	Glucose	0.35	0.01
5	Glucose	0.36	1.01
6	Motion	0.08	1.45
7	EMG	0.06	0.98
8	Blood Pressure	0.37	1.27
9	Pulse OXIMETER	0.4	1.01
10	Lactic acid	0.22	0.91
11	Accelerometer	0.45	0.45
12	Accelerometer	0.5	0.5
13	Respiratory	0.15	0.5
14	Pressure	0.15	0.45
15	Pressure	0.25	0.17
16	SDN-controller	0	0

3.1 Determining the Number of Clusters

To determine the number of clusters needed for our proposed algorithm, we have recorded the average end-to-end (E2E) delay (ms) for a different number of clusters ranging from 1 to 6 clusters. Clustering the network minimizes the E2E delay for a certain number of clusters; however, this is not always the case as maintaining network topology and exchanging the global network state causes an increase in

delay. Therefore, the delay increases when an excessive number of clusters is used. Figure 26 shows the delay for a different number of clusters. The E2E delay achieves its minimum value of 4.3 ms when three clusters exist before rising to 4.4 ms when four clusters exist; therefore, we will use three clusters for the rest of our work.

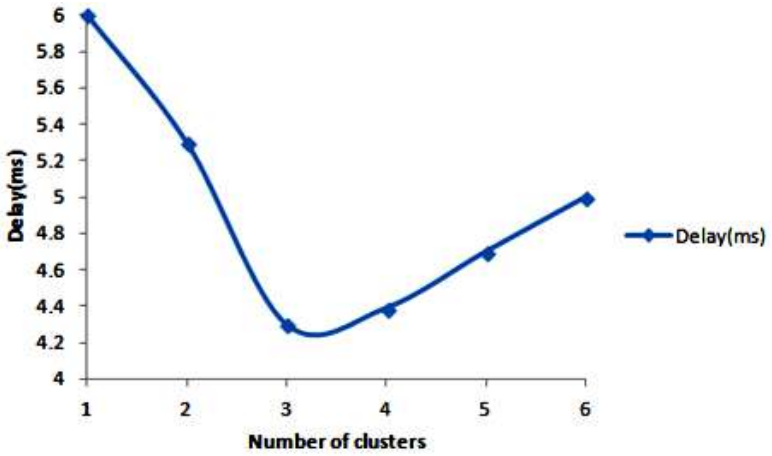

Figure 26: Delay for Different Number of Clusters

3.2 Estimation of the Energy Consumption Model and Priority-Based Transmission Rate

In WBAN, the sensor nodes are equipped with low-power batteries; hence, energy is scarce since recharging them is impossible. Most energy consumption is mainly due to transmitting the data, receiving the data, sensing the data, and collecting the data [128], [124]. However, data transmission is the most major cause behind energy drainage, and so is the bottleneck of energy consumption. There are many energy models in Literature, and the commonly used model is the first-order energy model [129]; hence, we are adopting this model in our research study. For every sensor node i in a cluster u, the SDN controller assigns T_i TDMA time slots. Equation 24 defines the needed energy for transmitting k bits at a distance d from the receiver.

$$E_{Tx(k,ni,d)}=\sum_{u=1}^{\alpha}\sum_{i=1}^{s_u}(E_{TX_{i,u}} * k + k * \eta * {d_i}^{\eta} * E_{amp}) \quad (24)$$

where α is the number of clusters in the network, s_u is the number of sensors in cluster u; $E_{TX_{i,u}}$ is the transmission energy cost of node i in a cluster u during its time interval for transmitting a single bit, k is packet size, ŋ is the path-loss coefficient, Eamp is the energy required for amplifier circuit, and d_i is the distance between node i and receiver. The energy needed for receiving k bits is given by Equation (25)

$$E_{RX(k,\alpha)} = \sum_{u=1}^{\alpha} \sum_{i=1}^{s_u} (k * E_{RX_{i,u}}) \quad (25)$$

where $E_{RX_{i,u}}$ is the energy cost of sensor node i in cluster u for receiving a single bit during its time interval;

Wireless communication in WBAN is greatly affected by human movement, so body movement contributes attenuation in the radio model [127]; therefore, in our study, we have considered the path loss effect caused by such attenuation. The transceiver energy parameters are shown in Table 15. The total energy consumption is the sum of the transmission energy and the receiving energy denoted as E_{CO} in Equation (26). The expression for the network energy consumption for sending and receiving k bits during the time interval T [123] is given as:

$$E_{CO(k,\alpha,d)} = (E_{TX_{(k,\alpha,d)}} + E_{RX_{(k,d)}})) \quad (26)$$

We assume that the number of events (n) occurring during time interval T with arrival rate λ follows a Poisson distribution which is given by:

$$P(n,T) = \frac{(PTR*T)^n}{n!} * e^{-PTR*T} \quad (27)$$

where PTR is the priority-based transmission rate is given in Equation 28.

$$PTR = 2^{p-2} * \lambda \quad (28)$$

The estimated network energy consumption during a time interval is given by Equation 29:

$$E(E_{CO(k,\alpha,d))} = \sum_{t=1}^{T} \frac{(PTR*t)^n}{n!} * e^{-PTR*t} * (E_{TX_{(k,\alpha,d)}} + E_{RX_{(k,d)}}) \quad (29)$$

A high priority value (p=2) is given to ED to be routed to the controller, and a low priority value (p=1) is given to ND. Achieving reliability is done by assigning a high transmission rate to ED and a low transmission rate to ND. This ensures that ED to be transmitted at full rate and ND is transmitted at half rate.

3.3 Path-Loss Model

The difference between energy consumption to transmit and receive data is referred to as path loss; in other words, it is the difference between the transmitted and received power represented in decibels (dBs) which is different from the loss or gain of antenna's radio signals. Besides, the path loss is affected by human movement; therefore, in WBAN, path loss occurs due to movement of the body, clothes, hands, and human postures. The posture of the human body affects electromagnetic signals. As a result, the path loss shows different behaviors along with different body parts. There are different path loss models in Literature. We included in our scheme the path loss model described in [130], which is a function of distance and frequency [131] shown in Equation 30.

$$PL = PL_0 + 10.\eta.\log10\ (d/d_0)+\sigma s \quad (30)$$

where PL0 is the path loss at reference distance d0 and ŋ is the path-loss coefficient.

The distance between the transmitter and the receiver is d, and σs is the shadowing factor that follows the Gaussian distribution of random variables [130]. The LOS path loss coefficient's value varies from 3 to 4, while for NLOS, its value is between 5 and 7.4. In our simulation, we used a fixed frequency (f) of 2.4 GHz, ŋ and σs as

3.38 and 4.1 [132], respectively. The path-loss is rewritten and shown in Equation 31.

$$PL = 10\log_{10}\left(\frac{(4\pi df)^2}{c}\right) \quad (31)$$

where c speed of light and *f* is frequency;

3.4 Energy-Efficient Routing (ER) Algorithm

Network partitioning is being adopted in many research studies to elongate the network lifetime. Hence, a Clustering-based routing algorithm is considered an optimum approach for network management [133]. The main advantages of the clustering mechanism are enhancing network lifetime and balancing the load [134]. For this purpose, we have adopted the clustering mechanism for our routing scheme.

An energy-efficient cluster head selection algorithm mainly focuses on balancing the energy consumption among nodes; specifically, a high transmission rate for ED depletes the sensor node's energy. We believe that maintaining balanced energy consumption among sensor nodes will enhance the network lifetime, throughput, and network stability. The initial energy is the same for all nodes, i.e., $E_{initial}$ = 0.5J. The nodes sense the vital parameters of the human body and send data to their associated CH. CHs send data to sink. Once the residual energy of CH is below a threshold (0.1 J), the controller assigns new CH by running GA to avoid packet loss, misconnection between sensors and CH, and to balance the energy among nodes. The residual energy of sensor node i is given by Equation 32.

$$E_{residual_i} = E_{initial_i} - E_{co_i} \quad (32)$$

where $E_{initial_i}$ is the initial energy of node i, and the difference of energy consumption of node i between current time interval T_i and one previous time

interval $T_i^{'}$ is considered so that the depleted energy node will not be considered for CH selection. The estimated consumption energy of node i is denoted as E_{co_i}. Since the energy consumption depends on the source-destination distance and source-destination channel attenuation, the signal-to-noise ratio (SNR) is included in the first objective function. We define the first and second objective functions of Ƒ as f_1, f_2 as Equations 33 and 34.

$$f_1 = \frac{E_{residual_i}^{T_i}}{\left(Eco_i^{T_i} - Eco_i^{T_i'}\right)} * \left(\frac{SNR}{\eta}\right)^{-1} \quad (33)$$

s.t. $E_{residual} \geq E_{min}$; where E_{min} is the minimum energy needed to be elected as CH;

$$f_2 = \left(\frac{d_{sensor\ to\ controller}}{\bar{d}_{sensors\ to\ controller}} * PL\right)^{-1} \quad (34)$$

where d is the distance between node and controller, and $\bar{d}$ is the average distance of nodes and controller, PL is the path-loss given in Equation (31). Therefore the fitness function is given by Equation 35.

$$Ƒ = \max\ (\omega_1 f_1 + \omega_2 f_2) \quad (35)$$

$$s.t\ \omega_1 + \omega_2 = 1;\ 0 \leq \omega_1 \leq 1,\ 0 \leq \omega_2 \leq 1$$

In the beginning, the controller runs the GA to determine the number of clusters to be three cluster heads. Then we have adopted the k-mean method [135] to cluster the nodes among these three clusters. The sensor node with maximum residual energy, minimum consumption energy difference in consecutive time intervals, and minimum distance to the controller will be chosen as CH. The same pseudo-code of GA is used in Table 2; however, the fitness function is Equation 35.

3.5 Scheduling

Based on the Time Division Multiple Access (TDMA) approach, our proposed system model uses a single-hop communication between sensors and cluster heads to maximize the network life and single or double-hop communication between cluster heads and the SDN controller. Packet collisions, idle listening, and overhead communication are reduced using a TDMA strategy that is better suited to the WBAN system. The scheduled TDMA access mechanism is based on the beacon mode IEEE 802.15.6 with superframe boundaries [130]. In the beginning, the controller divides the overall interval time T into10 ms length slots and assigns a one-time slot for every sensor for intra-cluster communication and inter-cluster communication. CHs use the same frequency band and transmit their data in different time slots to avoid collisions. As stated before, high priority is given to ED (p=2) to be routed to the controller, and low priority is given to ND (p=1). Hence, high priority data is assigned a sending interval less than that of ND. The priority-based sending interval (PS) is given by Equation (36):

$$PS = 2^{1-p} * T_i \tag{36}$$

Therefore, ED is transmitted with full rate and half the initial sensing interval to ensure delay-free transmission, whereas the ND is transmitted with half rate and full sensing interval.

Adjusting the transmission and receiving powers at sensor nodes is adopted from [136], where transmitting power should also be adjusted to meet the transmission rate assigned to the sensor node. The adjusted transmitting power is shown in Equation (37).

$$P^{TX} = \psi * E_{Tx(k,\alpha,d)} \tag{37}$$

$$\psi = PTR/k \tag{38}$$

where PTR is priority-based transmission rate given in Equation 28. The adjusted received power is shown in Equation (39)

$$P^{RX} = \psi * E_{RX(k,\alpha)} \quad (39)$$

where $E_{RX(k,\alpha)}$ is given in Equation (25);

3.6 ERQTM Model

The energy-efficient routing and QoS traffic management scheme comprises two algorithms: the energy-efficient routing (ER) algorithm illustrated by efficiently balancing the energy among sensor nodes and optimally selecting CHs. The second algorithm is the QoS traffic management (QTM) algorithm. Before going into these two algorithms' details, let's have a deeper view of the controller's control plane. Since the SDN controller manages the network, Figure 27 shows the controller's five main functional blocks.

Block 1: Determine the number of clusters: As stated in section 3.1, the controller determines the number of clusters needed for our algorithm.

Block 2: Run GA to optimally select the CH for each cluster: This is explained in section 7.6

Block 3: The controller determines the network topology and sends the flow tables to the sensor nodes.

Block 4: The controller, as stated in section 3.5, will assign the time slots for transmission and will manage the traffic flow by prioritizing the ED for transmission

Block 5: The controller, based on traffic flow, will assign transmission rate and sending interval and adjust power to the sensor nodes.

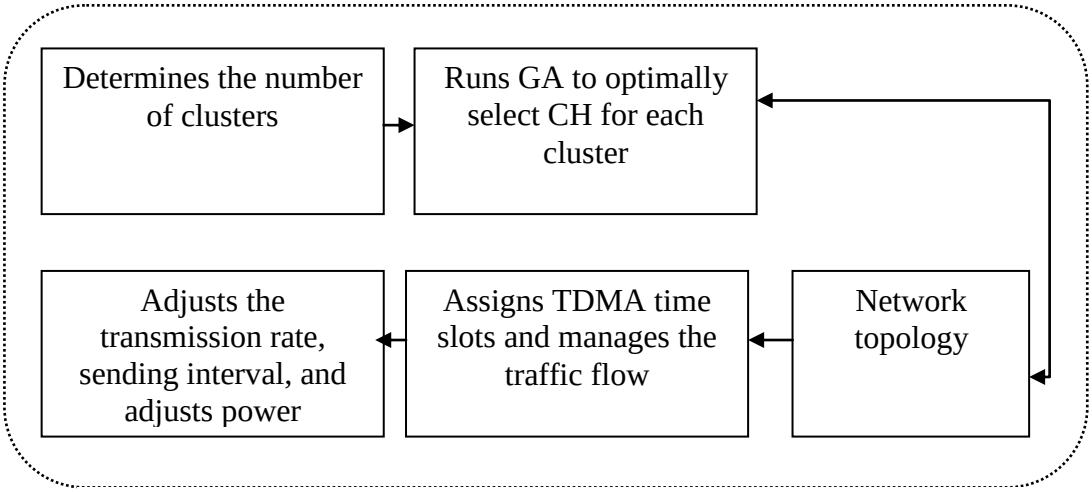

Figure 27: Controller's Main Blocks

Table 19 describes the QoS traffic management algorithm that precisely focuses on prioritizing the emergency data. Accordingly, the controller assigns the priority value, data rate, sending interval, and transmission power to the sensor node issuing the ED.

Table 19: QTM Algorithm

While data flow is available **do**
Data flow arrive
Check the type of data flow
If data.type=='emergency data' **then**
    assign the PTR value as in Equation 28 with p=2
    assign sending interval PS as in Equation 36 with p =2
    assign transmitting power as in Equation 37
Else
Assign the PTR value as in Equation 28 with p=1
Assign sending interval PS as in Equation 36 with p =1
Assign transmitting power as in Equation 37
End if
End While

The detailed description of the flowchart of the ERQTM scheme is as follows:

At the controller

When a packet arrives from CH to the controller, the following happens:

1. The controller checks the packet. If the packet is normal data and no beacon alarm is issued from any CH, then the controller adds normal data to its buffer (ND-buffer) for sending it later to the access point (AP).

2. However, if a beacon alarm has arrived at the controller, it could be either:

a. Emergency data alarm (ED): The controller will notify the CH, sending the ND to pause and store the rest of ND in CH's buffer (ND-buffer). The controller will assign a timeslot to the CH issuing the alarm beacon. Then CH will start sending its ED, and the controller will store it in its buffer (ED-buffer) to packetize it and send it immediately to AP.

b. Energy depletion of CH: The controller will run GA to select a new CH instead of the energy-depleted CH and will notify the CH that wants to send ND to send its data.

Whenever the controller finishes either case a or b and no alarm has been issued, it will check if any waiting ND in the buffer of any CH. If so, the controller notifies the CH, which has paused sending its ND to resume sending its ND. The controller will store the ND in its ND buffer for later transmission to AP and ends the algorithm.

3. If the packet is ED, the controller will immediately accept it. If during sending the ED, the controller receives an alarm:

a. Emergency Data beacon alarm: it will notify the CH that is issuing the ED alarm beacon to store it in its ED buffer. Once the controller finishes receiving the current

emergency data, it will assign a timeslot for CH, issuing the alarm to send its ED. The controller then adds ED in its ED buffer.

b. Energy Depletion beacon alarm: the controller will run GA to select a new CH instead of the depleted CH.

The controller will keep checking if no waiting ED exists to end the algorithm; otherwise, it will either follow steps 3a or 3b.

Traffic flow at the cluster head:

1. Each CH keeps tracking its residual energy
2. If residual energy is less than E_{min} it will either send an alarm beacon to the controller if it is a single-hop away or nearest CH if two hops away from the controller. The nearest neighbor will send this alarm beacon to the controller
3. The controller will assign a new CH to replace the current CH after receiving the alarm beacon.
4. The controller will use GA for step 3

If CH receives an ED from the controller, it will do the following:

5. Notifies all its sensors to stop sending any packet
6. Stores, if available, any ED or ND in respective buffers: ED-buffer or ND-buffer and immediately acts accordingly to the controller's request
7. Once finished, it will resume sending any ED in buffer first, then ND-buffer
8. It will notify its sensors to send their data.
9. However, if it receives ED from its sensor during sending its ND to the controller, it will store the current ND in its ND buffer and sends ED to the controller, then resumes sending ND. And if it receives normal data, it will store the ND-buffer data to be sent later.

The above steps are shown in Figure 28 under QoS Traffic Management (QTM) flowchart.

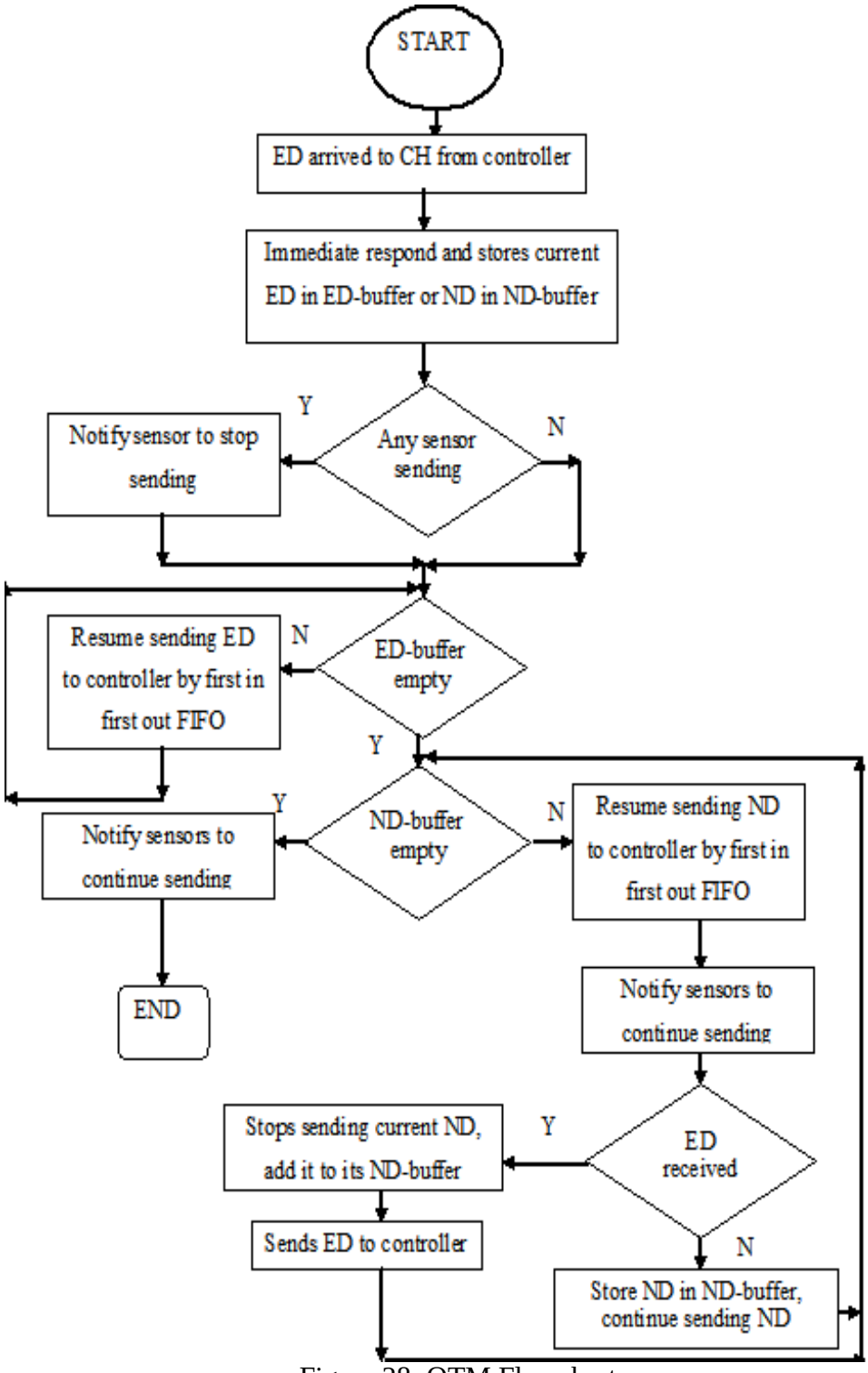

Figure 28: QTM Flowchart

Based on ER and QTM, the proposed flowchart of the ERQTM scheme is shown in Figure 29.

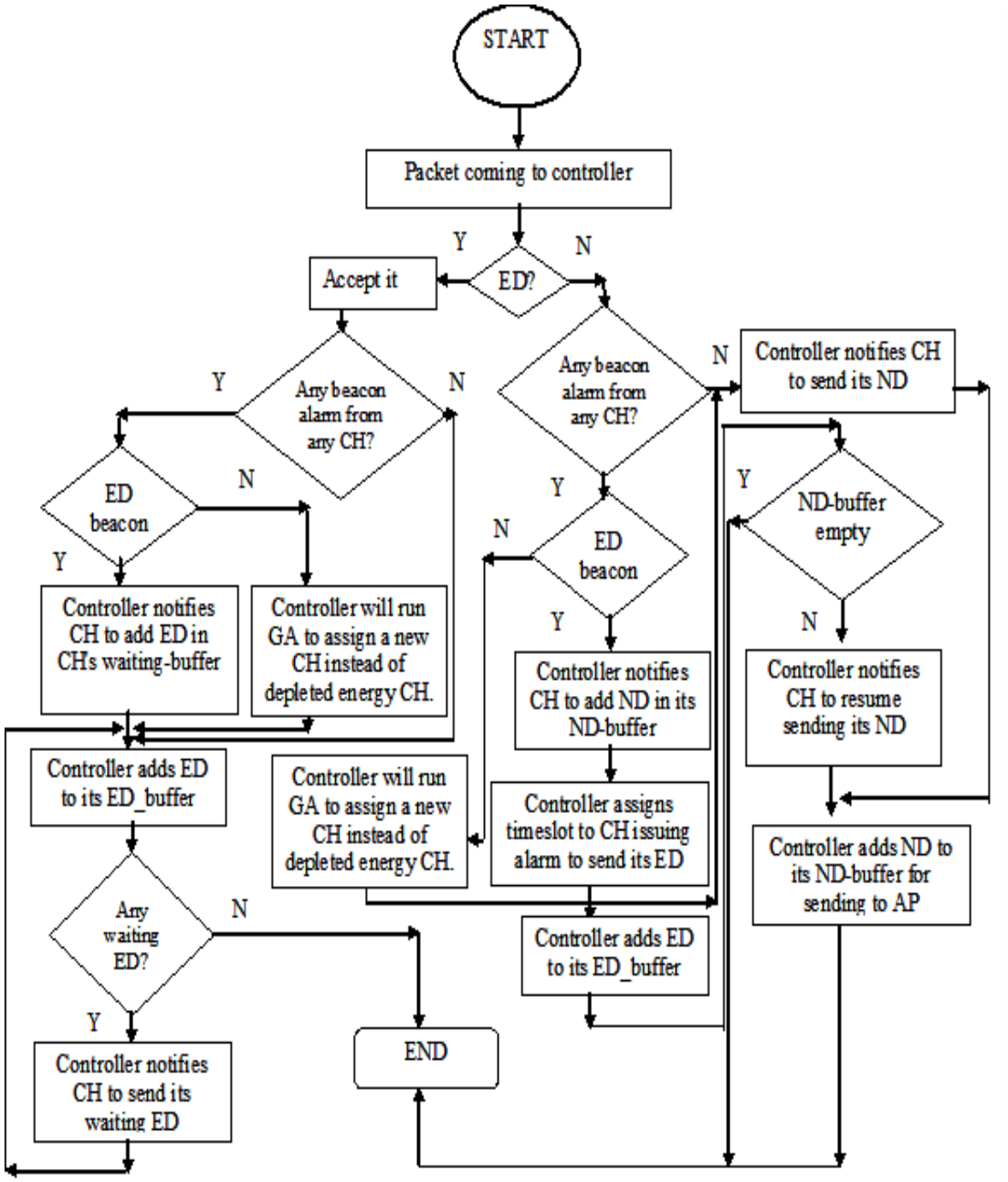

Figure 29: Flowchart of ERQTM Scheme

4. Performance Evaluation

Our proposed ERQTM algorithm has been implemented using MATLAB 2019. We have run our scheme 50 times for each round and took the average to achieve a 95% confidence interval. The performance verification and analysis have been carried out for Intra-Software Defined Wireless Body area network (Intra-SDWBAN) in a room with a patient having 15 sensor nodes and one SDN controller placed on his body as described in Section 3. We have selected three protocols from the state-of-the-art, namely, E-HARP [133], PERA [132], and PAP [115], for performance evaluation since they are relative to our proposed algorithm and most recent studies in the Literature of WBAN. The network performance metrics considered in comparison and are mostly used in WBAN Literature are as follows: end-to-end E2E delay, Throughput, Network lifetime and Stability, Residual energy, and computational complexity. Simulation parameters are provided in Table 20.

Table 20: Simulation Parameters for ERQTM Algorithm

Parameter	Value
Simulation area	$2x2m^2$
Number of sensor nodes	15
Position of sensors	Table 18
Duration of time slot	10 ms
Super-frames duration	700 ms
Packet size	1200 bit
Noise power	-94dBm
Transmission power range	-30dBm to 0 dBm
Coefficient of LOS	3.38
Coefficient of NLOS	5.9
Initial energy $E_{initial}$	0.5 J
E_{rmin}	0.1 J
E_{TX}	16.7 nJ/bit
E_{RX}	36.1 nJ/bit
E_{amp}	1.97 nJ/bit
Frequency	2.4 GHz
Initial packet transmission rate	5 packets/sec
Packet size	3000 bits
Wavelength	0.138m

4.1 Network Lifetime and Network Stability

In our performance evaluation, the network lifetime is considered the time it takes for the last node to die (LND), and the network stability is defined as the duration of time for the first node to die (FND). We have run our simulation for different rounds and recorded the number of rounds that the first node to die as network stability and the last node to die as network lifetime for the different protocols, as shown in Table 21 and Figure 30. As it can be seen, almost the same performance evaluation was recorded for our proposed scheme ERQTM and E-HARP with a very slight increase for the latter. This slight increase is due to the harvested energy that the authors in [133] have empowered the sensor nodes with energy to increase the network lifetime. Although we didn't consider any external harvest technique in our scheme, close results were recorded. The proposed energy-efficient routing algorithm positively impacts the overall network lifetime. Authors of PERA [132] have suggested placing the sensor nodes of emergency data like EEG, ECG close to the sink to save energy and the rest of the nodes use multi-hop communication to communicate with the sink. Therefore, the nodes close to the sink will frequently communicate other nodes' data to the sink, causing quick energy depletion. That is shown in Table 21. Authors of PAP [115] suggested dynamically allocating transmission rates to sensors that send emergency data to ensure reliability and delay-free communication. However, sending with high transmission rates by power-limited sensor nodes will lead to early death, especially if no alternative scheme is present to handle this issue. The FND is recorded at 5000, 4500, 3000, and 2200 rounds for ERQTM, E-HARP, PERA, and PAP, respectively. Moreover, the LND is recorded at 10000, 9000, 7000, and 5000 rounds for ERQTM, E-HARP, PERA, and PAP, respectively. The overall percentage

of average improvement of ERQTM over E-HARP, PERA, and PAP is almost 4%, 34%, and 56%, respectively.

Table 21: Network Stability and Network Lifetime Comparison

Protocol name	Number of alive nodes at different number of rounds									
	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
ERQTM	15	15	15	15	12	9	5	3	2	0
E-HARP	15	15	15	15	11	9	4	3	0	0
PERA	15	15	14	9	5	2	0	0	0	0
PAP	15	15	8	2	0	0	0	0	0	0

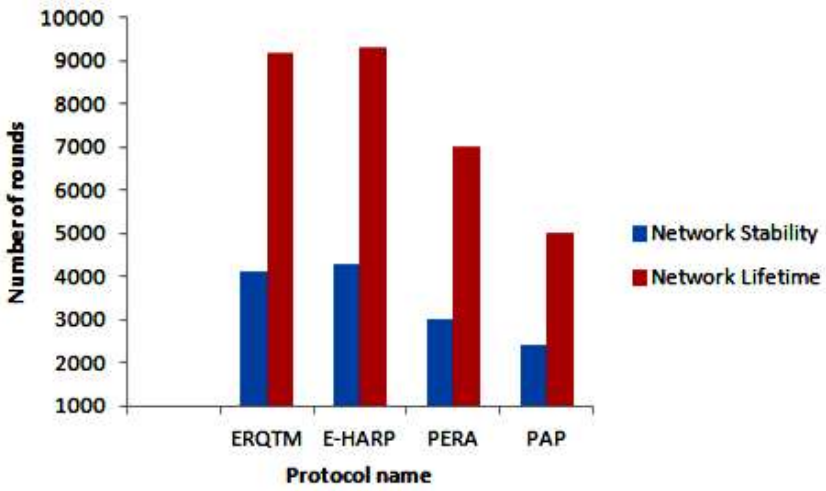

Figure 30: Network Stability and Network Lifetime Comparison

4.2 Network Throughput

Network throughput refers to successfully transmitted data in a defined time. A high throughput network is most desired in terms of performance. Figure 31 shows the proposed ERQTM throughput at high priority (p=2) and low priority (p=1) levels. Since data transmission rate is highly associated with emergency data or high priority

data; therefore, the system achieves higher throughput for high-priority data than low-priority data. That is illustrated in Figure 31.

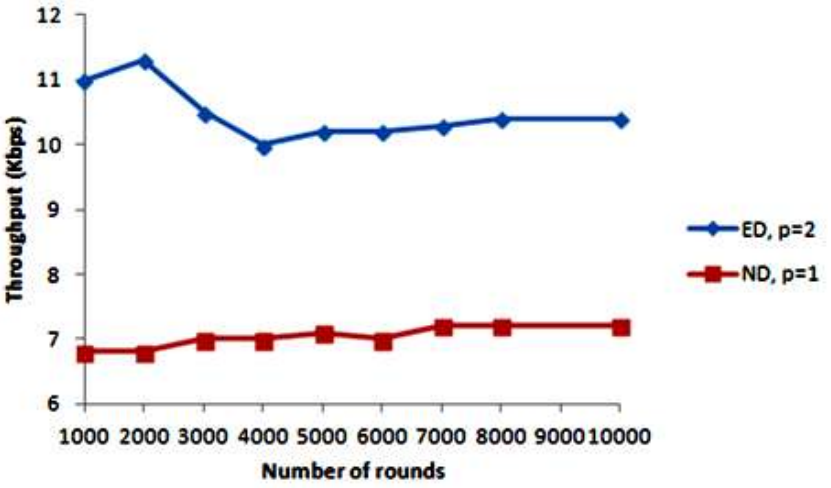

Figure 31: ERQTM Throughput for ED and ND

Figure 32 compares throughput values of our proposed ERQTM scheme with E-HARP, PERA, and PAP for a different number of rounds. As seen at the beginning of the simulation, PAP seems to outperform the E-HARP, PERA, and PAP protocols with a slight improvement compared to the ERQTM protocol. That is due to the high transmission rates assigned to ED, ensuring delay-free and successfully received packets. Also, no clustering methods are incorporated in our proposed algorithm and E-HARP that causes rigorous processing for the selection of CHs. Due to the quick depletion of residual energies, the throughput is negatively affected at 3000 rounds and above. In our scheme, the SDN controller is responsible for running GA to select CHs optimally; thus, we can see that this positively impacts saving the sensor nodes' residual energies and enhancing network throughput. Hence, it maintains its superiority compared with the others, and the overall average improvements of ERQTM over E-HARP, PERA, and PAP are almost 3%, 6%, and 7%, respectively.

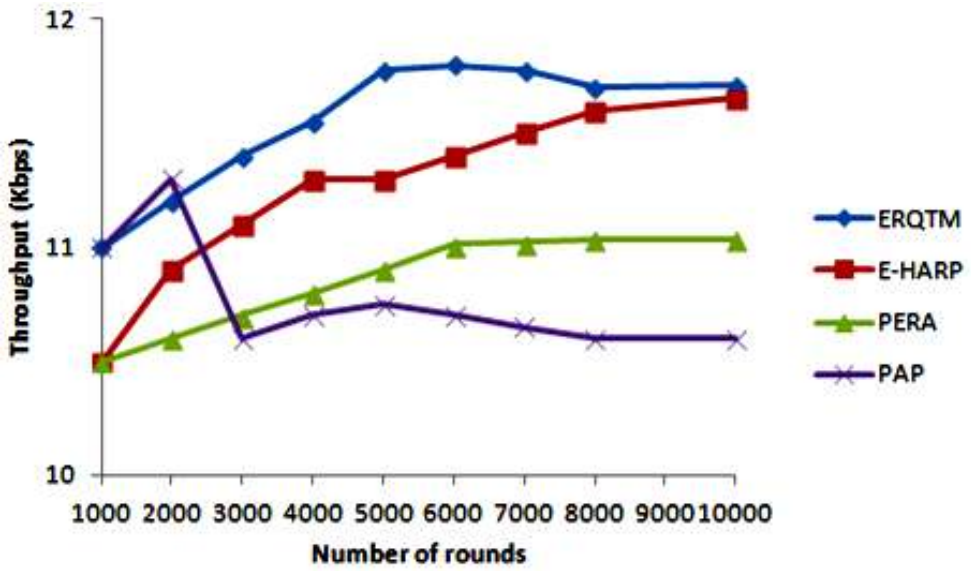

Figure 32: Throughput Comparison

4.3 Residual Energy

The total residual energy refers to the sum of all the sensor nodes' remaining energy levels in the network. The most energy consumption is due to transmission and receiving the data. The proposed energy-efficient routing algorithm optimally chooses CH considering the nodes' consumption rate, residual energy, distance to the controller. This shows a positive impact in increasing the network lifetime and, as a result, in saving energy. Figure 33 compares the residual energy of our proposed ERQTM with E-HARP, PERA, and PAP. Both ERQTM and E-HARP algorithms last more than 10000 rounds; however, in the proposed ERQTM scheme, an SDN controller's presence positively affects saving sensor nodes' energy where the SDN controller does all necessary calculations. That explains why the ERQTM scheme has more energy-saving than E-HARP. All in all, looking at the residual energy of all protocols, one can see that the average improvements of the ERQTM over E-HARP, PERA, and PAP protocols are almost 13%, 16%, and 31%, respectively.

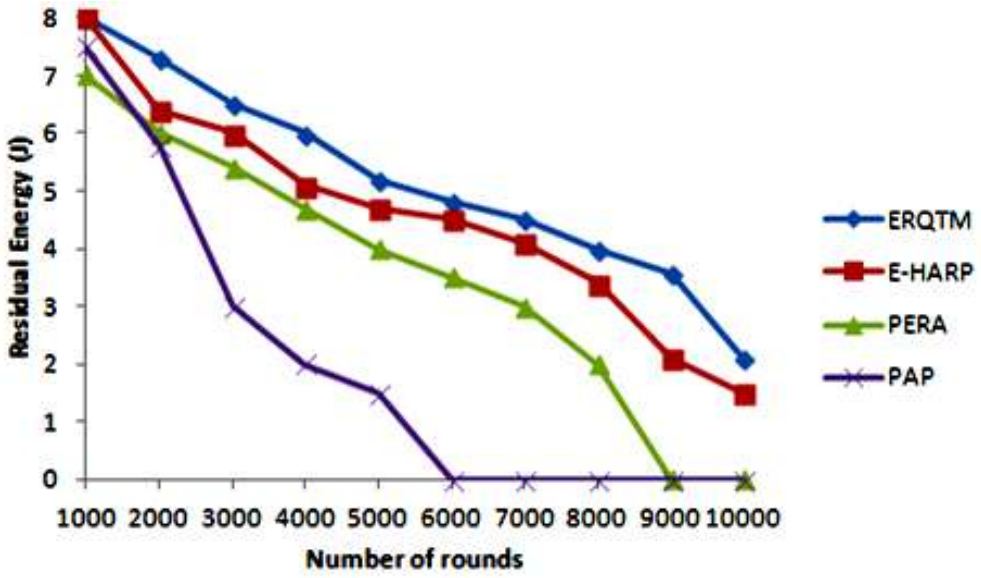

Figure 33: Residual Energy Comparison

4.4 End-to-End Delay

An important performance metric, especially for WBANs, is the end-to-end delay. We aim to have minimum delay specifically for emergency data to achieve high QoS. In the proposed ERQTM scheme, executions of routing algorithm and optimal CHs selection are done on behalf of the SDN controller, not the E-HARP case. In E-HARP, each sensor node does the necessary calculations and communicates with the sink. This will cause more delay caused by calculating and transmitting the data between sensors and the controller. In Figure 34, one can see that at the beginning, the ERQTM and E-HARP algorithms experience higher delays than other schemes. This delay is caused by the optimal selection of cluster heads, which is not the PAP and PERA case. After 2000 rounds, the ERQTM algorithm achieves minimum end-to-end delay among the rest of the schemes. The clustering mechanism proves to be a well-desired routing algorithm to achieve minimum delay. The overall average improvements of ERQTM over E-HARP, PERA, and PAP are almost 15%, 56%, and 51%, respectively.

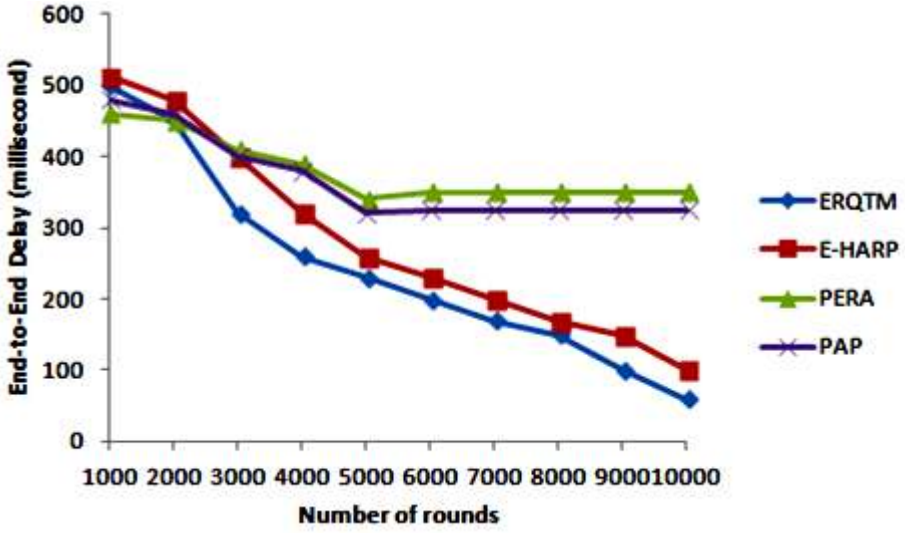

Figure 34: End-to-End Delay Comparison

4.5 Computational Complexity

A general important performance metric, especially for time-sensitive applications as e-Health applications, is the computational complexity refers to the computational time or CPU execution time. The simulations were performed on a PC with an Intel Core i5 CPU and 8 GB of RAM. Table 22 represents the comparison of the computational time (microseconds) of our proposed ERQTM algorithm with E-HARP, PERA, and PAP algorithms for a different number of sensor nodes. As it is seen, the computational time increases with the increase of sensor nodes. The computational time of PERA and PAP is less than that of ERQTM and E-HARP due to the clustering methods and the optimal cluster heads in ERQTM and E-HARP. That verifies the computational time's direct dependency on the network topology and the clustering method used.

Table 22: Computational Complexity Comparison (μs)

No.of sensor nodes	ERQTM	E-HARP	PERA	PAP
5	12	14	10	5
7	15	17	13	9
10	17	20	15	12
12	23	25	20	18
15	30	34	26	24

5. Discussions

An energy-efficient routing and QoS-supported traffic management for software-defined WBAN (ERQTM) is proposed in this research study. The proposed ERQTM scheme incorporates an energy-efficient routing algorithm based on a genetic algorithm to optimally select CH and considers various metrics such as nodes' residual energy, energy consumption rate, location, link quality, and signal-to-noise ratio path-loss effect. The second algorithm manages the traffic flow and prioritizes the emergency data once issued by any node (CH, cluster member, or even SDN controller). This algorithm ensures reliability by giving high priority for routing the emergency data and assigning transmission power accordingly. The overall scheme outperforms the compared protocols in terms of network stability, network lifetime, residual energy, throughput, and end-to-end delay. Since our proposed scheme is based on assigning a high transmission rate and transmitting power for ED transmission, then harvested energy along with the energy-efficient routing algorithm will be needed as future work to cease the drawback issue caused by such assignments. The proposed scheme can be extended for future work for considering cross-layer interactions for complex network scenarios, especially for COVID-19 patients. We have written a paper [137] based on the ERQTM algorithm.

Appendix B: Confidence Interval Estimation

In this part of the thesis, we have explained the 95 % confidence interval estimation for the latency values under the genetic algorithm when three controllers are present in the system for different faulty node percentages. We have applied the t-distribution [138] to verify the correctness of the obtained results. The table below includes the average latency values that are obtained from the outputs of 50 simulation runs.

Fault %	Latency
0	0.50664
1	0.5124
5	0.52322
10	0.52509
20	0.52489
50	0.52995

We calculated the 95% confidence interval for each percentage of faulty nodes, and the value for the obtained latency falls within the 95% confidence interval.

Case 1: 0% faulty nodes

The below table shows the 50 runs values for latency under 0% faulty nodes.

No. of runs	Latency
1	0.50419
2	0.50429
3	0.50439
4	0.50449
5	0.50459
6	0.50469
7	0.50479
8	0.50489
9	0.50499
10	0.50509

11	0.50519
12	0.50529
13	0.50539
14	0.50549
15	0.50559
16	0.50569
17	0.50579
18	0.50589
19	0.50599
20	0.50609
21	0.50619
22	0.50629
23	0.50639
24	0.50649
25	0.50659
26	0.50669
27	0.50679
28	0.50689
29	0.50699
30	0.50709
31	0.50719
32	0.50729
33	0.50739
34	0.50749
35	0.50759
36	0.50769
37	0.50779
38	0.50789
39	0.50799
40	0.50809
41	0.50819
42	0.50829
43	0.50839
44	0.50849
45	0.50859
46	0.50869
47	0.50879
48	0.50889
49	0.50899
50	0.50909
average	0.50664

The degrees of freedom is given by n-1 = 49, the 100(1-α) percent confidence interval is determined by:

$\bar{y}$ - $t_{\frac{\alpha}{2},(n-1)} \times \frac{s}{\sqrt{n-1}}$ < μ< $\bar{y}$ + $t_{\frac{\alpha}{2},(n-1)} \times \frac{s}{\sqrt{n-1}}$; where $\bar{y}$ is the average, s is the standard deviation, n is the number of runs. Accordingly, the confidence interval is determined by:

$$\overline{y} \mp t_{\frac{\alpha}{2},(n-1)} \times \frac{s}{\sqrt{n}}$$

The t-distribution value is given by: $t_{0.025,49}$ = 2.0115. The variance of the 50 runs sample is $s^2 = \sum_{i=1}^{n=50} \frac{(y_1 - \overline{y})^2}{n-1}$ = 2.125E-06; therefore the standard deviation s = $\sqrt{2.12E-06}$ = 0.0014577; and the half size of confidence interval is: b=$\frac{0.0014577}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is :

0.50664 $\mp$2.01155 $\times \frac{0.0014577}{\sqrt{50}}$, the obtained latency value 0.50664 is between the 95% confidence interval.

Case 2: 1% faulty nodes

The below table shows the 50 runs values for latency under 1% faulty nodes.

No. of runs	Latency
1	0.51
2	0.5101
3	0.5102
4	0.5103
5	0.5104
6	0.5105
7	0.5106
8	0.5107
9	0.5108
10	0.5109
11	0.511
12	0.5111
13	0.5112
14	0.5113
15	0.5114
16	0.5115
17	0.5116
18	0.5117
19	0.5118
20	0.5119
21	0.512

22	0.5121
23	0.5122
24	0.5123
25	0.5124
26	0.5125
27	0.5126
28	0.5127
29	0.5128
30	0.5129
31	0.513
32	0.5131
33	0.5132
34	0.5133
35	0.5134
36	0.5135
37	0.5136
38	0.5137
39	0.5138
40	0.5139
41	0.514
42	0.5141
43	0.5142
44	0.5143
45	0.5144
46	0.5145
47	0.5146
48	0.5147
49	0.5148
50	0.5149
average	0.5124

The variance of the 50 runs sample is $s^2 = \sum_{i=1}^{n=50} \frac{(y_1 - \overline{y})^2}{n-1}$ = 2.125E-06; therefore the standard deviation s = $\sqrt{2.125\mathrm{E}-06}$ = 0.0014577 ; and the half size of confidence interval is: b=$\frac{0.0014577}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is :

0.5124 $\mp 2.01155 \times \frac{0.0014577}{\sqrt{50}}$, the obtained latency value 0.5124 is between the 95% confidence interval.

Case 3: 5% faulty nodes:

The below table shows the 50 runs values for latency under 5% faulty nodes.

No. of runs	Latency
1	0.52298
2	0.52299
3	0.523
4	0.52301
5	0.52302
6	0.52303
7	0.52304
8	0.52305
9	0.52306
10	0.52307
11	0.52308
12	0.52309
13	0.5231
14	0.52311
15	0.52312
16	0.52313
17	0.52314
18	0.52315
19	0.52316
20	0.52317
21	0.52318
22	0.52319
23	0.5232
24	0.52321
25	0.52322
26	0.52323
27	0.52324
28	0.52325
29	0.52326
30	0.52327
31	0.52328
32	0.52329
33	0.5233
34	0.52331
35	0.52332
36	0.52333
37	0.52334
38	0.52335
39	0.52336
40	0.52337
41	0.52338
42	0.52339
43	0.5234
44	0.52341
45	0.52342
46	0.52343
47	0.52344

48	0.52345
49	0.52346
50	0.52347
average	0.52322

The variance of the 50 runs sample is $s^2 = \sum_{i=1}^{n=50} \frac{(y_1 - \overline{y})^2}{n-1}$ = 2.125E-08; therefore the standard deviation s = $\sqrt{2.125\mathrm{E}-08}$ = 0.0001458; ; and the half size of confidence interval is: b=$\frac{0.001458}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is :

$0.52322 \mp 2.01155 \times \frac{0.001458}{\sqrt{50}}$, the obtained latency value 0.52322 is between the 95% confidence interval.

Case 4: 10% faulty nodes

The below table shows the 50 runs values for latency under 10% faulty nodes.

No. of runs	Latency
1	0.52264
2	0.52274
3	0.52284
4	0.52294
5	0.52304
6	0.52314
7	0.52324
8	0.52334
9	0.52344
10	0.52354
11	0.52364
12	0.52374
13	0.52384
14	0.52394
15	0.52404
16	0.52414
17	0.52424
18	0.52434
19	0.52444
20	0.52454
21	0.52464
22	0.52474
23	0.52484
24	0.52494
25	0.52504
26	0.52514

27	0.52524
28	0.52534
29	0.52544
30	0.52554
31	0.52564
32	0.52574
33	0.52584
34	0.52594
35	0.52604
36	0.52614
37	0.52624
38	0.52634
39	0.52644
40	0.52654
41	0.52664
42	0.52674
43	0.52684
44	0.52694
45	0.52704
46	0.52714
47	0.52724
48	0.52734
49	0.52744
50	0.52754
average	0.52509

The variance of the 50 runs sample is $s^2 = \sum_{i=1}^{n=50} \frac{(y_1 - \overline{y})^2}{n-1}$ = 2.125E-06; therefore the standard deviation s = $\sqrt{2.125\text{E} - 06}$ = 0.001458 ; and the half size of confidence interval is: b=$\frac{0.001458}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is :

$0.52509 \mp 2.01155 \times \frac{0.001458}{\sqrt{50}}$, the obtained latency value 0.52509 is between the 95% confidence interval.

Case 5: 20% faulty nodes

The below table shows the 50 runs values for latency under 20% faulty nodes.

No. of runs	Latency
1	0.52244
2	0.52254
3	0.52264
4	0.52274
5	0.52284

6	0.52294
7	0.52304
8	0.52314
9	0.52324
10	0.52334
11	0.52344
12	0.52354
13	0.52364
14	0.52374
15	0.52384
16	0.52394
17	0.52404
18	0.52414
19	0.52424
20	0.52434
21	0.52444
22	0.52454
23	0.52464
24	0.52474
25	0.52484
26	0.52494
27	0.52504
28	0.52514
29	0.52524
30	0.52534
31	0.52544
32	0.52554
33	0.52564
34	0.52574
35	0.52584
36	0.52594
37	0.52604
38	0.52614
39	0.52624
40	0.52634
41	0.52644
42	0.52654
43	0.52664
44	0.52674
45	0.52684
46	0.52694
47	0.52704
48	0.52714
49	0.52724
50	0.52734
average	0.52489

The variance of the 50 runs sample is s2 =$\sum_{i=1}^{n=50} \frac{(y_1-\bar{y})^2}{n-1}$ = 2.125E-06; therefore the standard deviation s = $\sqrt{2.125E-06}$ = 0.001458

; and the half size of confidence interval is: b=$\frac{0.001458}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is : 0.52489 $\mp 2.01155 \times \frac{0.001458}{\sqrt{50}}$, the obtained latency value 0.52489 is between the 95% confidence interval.

Case 6: 50% faulty nodes

The below table shows the 50 runs values for latency under 50% faulty nodes.

No. of runs	Latency
1	0.5275
2	0.5276
3	0.5277
4	0.5278
5	0.5279
6	0.528
7	0.5281
8	0.5282
9	0.5283
10	0.5284
11	0.5285
12	0.5286
13	0.5287
14	0.5288
15	0.5289
16	0.529
17	0.5291
18	0.5292
19	0.5293
20	0.5294
21	0.5295
22	0.5296
23	0.5297
24	0.5298
25	0.5299
26	0.53
27	0.5301
28	0.5302
29	0.5303
30	0.5304
31	0.5305
32	0.5306

33	0.5307
34	0.5308
35	0.5309
36	0.531
37	0.5311
38	0.5312
39	0.5313
40	0.5314
41	0.5315
42	0.5316
43	0.5317
44	0.5318
45	0.5319
46	0.532
47	0.5321
48	0.5322
49	0.5323
50	0.5324
average	0.52995

The variance of the 50 runs sample is s2 =$\sum_{i=1}^{n=50} \frac{(y_1-\bar{y})^2}{n-1}$ = 2.125E-06; therefore the standard deviation s = $\sqrt{2.125E-06}$ = 0.001458 ; and the half size of confidence interval is: b=$\frac{0.001458}{\sqrt{50}} t_{0.025,49}$. Thus the 95% confidence interval is :

0.52995 $\mp 2.01155 \times \frac{0.001458}{\sqrt{50}}$, the obtained latency value 0.52995 is between the 95% confidence interval.

Appendix C: Simulation Code

% Nivine Samarji , nivine.samarji@emu.edu.tr

Appendix C1: The code of the FTMBS Scheme

```
clear all
close all
clc
%%%%%%%%%%%     Optimization parameters
n_cont_des=input('please enter number of desired controllers\n');                  %
initializing number of desired controllers
EN=2;                           % initializing energy consumed for transmission

%%%%%%%%%%
%/////////////// user parameters ///////////////////

nodes=500;                      % initializing number of nodes
area=[200 200];                 % initializing area of deployment
BS=area/2;                      % initializing base station location

Efs=10e-12;                     % initializing comunication constant values as per base
paper
Ems=0.0013e-12;                 % initializing comunication constant values as per base
paper
Eelct=50e-9;                    % initializing comunication constant values as per base
paper
L0=4000;
L=L0;                           % initializing comunication constant values as per base
paper (packets)
E0=0.5;                         % initializing comunication constant values as per base
paper (initial energy)
Eda=5e-9;                       % initializing comunication constant values as per base
paper (data aggregation)
d0=88;                          % initializing comunication constant values as per base
paper (distance threshold)
% fault_percentage=[0.01;0.05;0.1;0.2;0.3;0.4;0.5;0.6;0.7];
fault_percentage=[0.05e-2];
t_max=300;
T_treshold=2.63e-3;
miu=3000;
L_treshold=2600;
w=[25/48,13/48,7/48,3/48];
FND_GA=[];
LND_GA=[];
FND_GRASP=[];
LND_GRASP=[];

notations={'sb','sc','sr','sg','sm','sy','sk','ob','oc','or','og','om','oy','ok','db','
dc','dr','dg','dm','dy','dk','*b','*c','*r','*g','*m','*y','*k'};
% clusters color and marker style

%/////////// deplying nodes ////////////////////

for T=1:2:3                                                          % T=1 without
sdn, T=2 with SDN
    nodes_alive=[];
    flag_2=0;
    node_locs=[];                               % initializing node locations
    while(size(node_locs,1)<nodes)              % deployment till we deploy all nodes
        for i=1:ceil(2*(area(1)*area(2)/nodes)^0.5):area(1)-1                    %
deployment grid wise row with 20x20
            for j=1:ceil(2*(area(1)*area(2)/nodes)^0.5):area(2)-1                %
deployment grid wise colm with 20x20
```

```
                y=randi([i i+ceil(2*(area(1)*area(2)/nodes)^0.5)-1],1,1);          %
randomly taking node position X
                x=randi([j j+ceil(2*(area(1)*area(2)/nodes)^0.5)-1],1,1);          %
randomly taking node position Y

                node_locs=[node_locs;x y];      % saving node in node locations array
                if(size(node_locs,1)==nodes)    % break the deployment if we deploy all
nodes
                    break
                end
            end
            if(size(node_locs,1)==nodes)        % break the outerloop if we deploy all
nodes
                break
            end
        end
    end

    %///////// predicting alternative paths for sdn ///


    for ii=1:nodes
        curr_node=node_locs(i,:);                           % saving alternative paths
for every node in sdn
        dis=sqrt( (node_locs(:,1)-curr_node(1)).^2 + (node_locs(:,2)-curr_node(2)).^2);
        com=[(1:nodes)' dis];
        com=sortrows(com,2);
        nodes_table{ii}=com(2:end,:);
    end

    %////////////////////////////////////////////

    figure(1)                                       % fiugre handler
    plot(node_locs(:,1),node_locs(:,2),'ok','MarkerFaceColor',[1,0,0])  % plotting all
nodes
    hold on
    plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20)  %
plotting base station location
    axis([0 area(1) 0 area(2)+60]);             % making axis limits to show all nodes
and base station
    hold off

    %////////////////////// Similarity Graph prediction /////////////////

    X=node_locs;                                    % copying node locations in X variable
    Y=node_locs;                                    % copying node locations in Y variable
    E = pdist2(X,Y);                                % predicting elucidian distance of
every node to other node
    E = E / max(E(:));                              % normalizing distances with in range
[0 1]

    sigma=0.3;                                      % taking sigma as 0.3
    for i=1:size(E,1)                               % calculating similarity matrix
        for j=1:size(E,2)
            if ~(i==j)                              % as per the equation (5) from base
paper
                W(i,j)=exp(-1*(E(i,j)^2/(2*sigma^2)));
            else
                W(i,j)=0;
            end
        end
    end

    Adj_mat=W;          % copying W variable to adjacency matrix
```

```
    Deg_mat(1:size(Adj_mat,1),1:size(Adj_mat,2))=0;     % initilizing degree matrix
    for i=1:size(Adj_mat,1)         % copying row wise to calculate degree matrix as
per the base paper
        Deg_mat(i,i)=sum(Adj_mat(:,i));
    end
    Lap_mat=Deg_mat - Adj_mat;      % calculating laplacian matrix as per the base
paper L=D-A

    for i=1:length(Lap_mat)
        normalised_Lap_mat(:,i) = Lap_mat(:,i) / Lap_mat(i,i);      % making laplacian
matrix to normalized laplacian
    end

    d_to_BS=sqrt((node_locs(:,1) - BS(1)).^2 + (node_locs(:,2) - BS(2)).^2);        %
predicting distances of all nodes to Base station
    d_to_BS_avg=(mean(d_to_BS)^2);                                                  %
taking average of distances

    K=round((sqrt(nodes)/sqrt(2*pi)) * sqrt(Efs/Ems) *
(sqrt(area(1)*area(2))/(d_to_BS_avg)));    % predicting K as per base paper


    %/////////// clustring using k means ////////////////////////////////

    for i=1:length(d_to_BS)                                                         %
calculating D as per the base paper
        D(i,i)=d_to_BS(i);
    end

    A=(D*-1/2) * Adj_mat * (D*-1/2);                                                %
A'=D1/2 A D-1/2
    [VL,D1] = eig(A');                                                              %
calculating eigen values VL

    figure(2)
    idx = kmeans(VL(:,1:K),K);
    colors=distinguishable_colors(K);
        for i=1:K
            [r,c]=find(idx==i);
% loop to check which node belongs to which cluster decided by k means
            plot(node_locs(r,1),node_locs(r,2),'s','color',colors(i,:))
% ploting  the respective nodes with predefined color and marker style
            hold on
        end
        plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20)
% plotting base station location
        axis([0 area(1) 0 area(2)+60]);
        hold off
        title('Clustering results of the KSCA-WSN algorithm')
        drawnow


    %////////// communication phase ////////////////////////////////////

    node_locs(:,3)=d_to_BS;                                         % saving base
station distances at 3rd colm in node locations array
    node_locs(:,4)=idx;                                             % saving cluster
number at 4th colm in node locations array
    node_locs(:,5)=E0;                                              % saving initial
energies at 5th colm in node locations array
    node_locs(:,6)=1:nodes;                                         % saving id number
at 6th colm in node locations array
    node_locs(:,8)=0;
```

```
    for i=1:size(node_locs,1)                                          % loop to predict
transmit energies for all nodes to base station
        if(d_to_BS(i)<d0)
            Etx(i,1)=(L * Eelct) + (L * Efs * d_to_BS(i).^2);       % predicing as per
equation 2 in base paper
        else
            Etx(i,1)=(L * Eelct) + (L * Efs * d_to_BS(i).^4);
        end
    end

    controller=[];                    % id of controllers
    which_clus_no_contr=ones(K,1);
    cont_fault_flag=0;
    cont_fault_id=[];
    controllers_load=zeros(nodes+1,t_max);
    flag_cont=0;
    BSAWF_time=[];

    for i=1:t_max    % loop to run for 300 secs

        t_rate=(ceil(i/10)*100)/nodes;
        L=L0*t_rate;

        Erx(1:nodes,1)=L * Eelct;         % predicing reciving energy for all nodes
        Eagg(1:nodes,1)=L * Eda;          % predicing data aggregation energy for all
nodes

        %//////// cluster head selection /////////////////
        for j=1:K                                                      % loop to predict
cluster heads for all clusters
            [r,~]=find(node_locs(:,4)==j);                             % predicing nodes
in jth cluster
            cluster_nodes=node_locs(r,:);                              % copying
respective nodes in cluster_nodes array
            Sk=size(cluster_nodes,1);                                  % number of nodes
belongs to jth cluster Sk
            if which_clus_no_contr(j)==1
                for jj=1:size(cluster_nodes,1)                             % loop to
preict Ermin as per base paper equation (7)
                    Ermin(jj,1) = abs(Sk) * ((Erx(r(jj)) + Eagg(r(jj))));

                    if(cluster_nodes(jj,5) > Ermin(jj))         % in respective node is
above the Emin make the node as cluster head selection process
                        cluster_nodes(jj,7)=1;                  % making 1 will make
the node to enter in cluster head selection process
                    else
                        cluster_nodes(jj,7)=0;                  % 0 will make the node
not to enter in CH selection process
                    end
                end
                [r1,~]=find(cluster_nodes(:,7)==1);            % checking the nodes
which qualifies CH selection process
                if ~(isempty(r1))
                    ch_members=cluster_nodes(r1,6);
                    cluster_heads(j,1)=ch_members(randi([1 size(ch_members,1)],1,1));
% as per the base paper choosing randomly CH from qualified CH nodes

                    cluster_nodes_t=cluster_nodes;

cluster_nodes_t(find(cluster_nodes_t(:,6)==cluster_heads(j,1)),:)=[];
                    simin=find(cluster_nodes_t(:,5)==0);
                    cluster_nodes_t(simin,:)=[];
                    ch_members_mat{j}=cluster_nodes_t(:,6);
                else
```

```
                    ch_members_mat{j}=[];                       % otherwise if no CH
node qualifies, make the CH for that cluster to NaN
                    cluster_heads(j,1)=1.5;                     % a flag to show that
there is no CH in the cluster
                    node_locs(cluster_nodes(:,6),5)=0;
                end
            end
            cluster_nodes_t_id=ch_members_mat{j};
            cluster_nodes_tt=node_locs(cluster_nodes_t_id,:);
            simin=find(cluster_nodes_tt(:,5)==0);
            cluster_nodes_tt(simin,:)=[];
            ch_members_mat{j}=cluster_nodes_tt(:,6);
        end
        remain_CH=cluster_heads;
        remain_CH(find(remain_CH==1.5))=[];

        %%%%%%%%%%%%%%%%%            FFFFFFFFFFFFFFAAAAAAARRRRRRRRRRRnaz
controller selection
        n_cont=n_cont_des-length(controller)+flag_cont;

        for javad=1:K
            CH_node(javad,1)=length(ch_members_mat{javad});
        end

        %//////  evaluating number of alive nodes in each cluster  ////////

        c_faulty_node=zeros(K,1);
        for ezat=1:K
            c_members=ch_members_mat{ezat};
            for rajab=1:size(c_members,1)
                if node_locs(c_members(rajab),8)==1
                    c_faulty_node(ezat,1)=c_faulty_node(ezat,1)+1;
                end
            end
            CH_alive_node(ezat,1)=CH_node(ezat,1)-c_faulty_node(ezat,1);
        end

        %////////////////////////////////////////////////////////////////

        CH_not_controller=remain_CH;
        for naghme=1:length(controller)
            CH_not_controller(find(CH_not_controller==controller(naghme)))=[];
        end

        is_CH_fualty=zeros(length(CH_not_controller),1);

        if n_cont>0
            if size(CH_not_controller,1)>n_cont
                if T==1
                    tic

[cont_opt,CHrem,cost_opt,iteration,for_plot,meann]=GA_SDN(n_cont,CH_not_controller,node
_locs(CH_not_controller,1:2),is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node,
t_rate,EN,BS,w);
                    time_opt(T)=toc;
                elseif T==3
                    tic

[cont_opt,CHrem,cost_opt]=GRASP_SDN(n_cont,CH_not_controller,node_locs(CH_not_controlle
r,1:2),is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node,t_rate,EN,BS,cluster_h
eads,w);
                    time_opt(T)=toc;
                end
            else
                cont_opt=CH_not_controller';
```

```
            end
            if i==1
                controller=cont_opt;
            else
                ema=1;
                for suzan=1:length(cont_opt)
                    controller(cont_delete(suzan))=cont_opt(suzan);
                end
            end
            cont_opt=[];
        end

        if i==1
            time_opt_t=time_opt;
            if T==1
                optimalcost(T)=cost_opt;
                iteration_optimal=iteration;
                figure(10)
                plot(for_plot)
                set(gca,'FontSize',20)
                set(gca, 'FontName', 'Times new roman')
                title(strcat('minimum cost value vs iteration for GA'))
                xlabel('Iteration')
                ylabel('minimum cost')

                figure(11)
                set(gca,'FontSize',20)
                set(gca, 'FontName', 'Times new roman')
                plot(meann)
                title(strcat('average cost value vs iteration for GA'))
                xlabel('Iteration')
                ylabel('average cost')
            elseif T==3
                optimalcost(T)=cost_opt;
            end
        end
        %%%% clustring CH among controllers

        if ~isempty(controller)
            for asghar=1:n_cont_des
                CH_not_controller(find(CH_not_controller==controller(asghar)))=[];
            end
        end
        which_clus_no_contr=ones(K,1);
        which_clus_no_contr(node_locs(controller,4),1)=0;

        if ~isempty(CH_not_controller)

CH_cont_id=CH_cont_id_fun(node_locs(CH_not_controller,1:2),[node_locs(controller,1:2);B
S],[controller';0]);
        end

        %////////////////////////  load calculation   ////////////////////
        CH_load=t_rate*CH_alive_node;
        Con_load=[];
        for ezat=1:length(controller)
            Con_load(ezat,1)=CH_load(find(cluster_heads==controller(ezat)));   %
initializing controller load matrix
        end
        Con_load(length(controller)+1,1)=0;

        controller_t=[controller 0];
        if ~isempty(CH_cont_id)
            for zzz=1:length(controller_t)
                cont_data=[];
```

```
                if ~isempty(CH_not_controller)
                    for sss=1:size(CH_not_controller,1)
                        if CH_cont_id(sss)==controller_t(zzz)
                            Con_load(zzz,1)=Con_load(zzz,1)+CH_load(sss,1);    %
calculating controllers load
                            cont_data(1,1)=sss;
                            cont_data(1,2)=CH_not_controller(sss,1);
                            cont_data(1,3)=CH_load(sss,1);
                            cont_data(1,4)=controller_t(zzz);
                        end
                    end
                end
                cont_data_mat{zzz}=cont_data;
            end
        end

        %///////////////  latency calculation  //////////////////////////

        lambda=Con_load;
        Con_loc=[node_locs(controller,1:2);BS];
        CH_loc=node_locs(CH_not_controller,1:2);
        De2e=zeros(length(controller_t),1);

        if ~isempty(CH_cont_id)
            for zahra=1:length(controller_t)
                if ~isempty(CH_not_controller)
                    if zahra~=length(controller_t)
                        if node_locs(controller(zahra),8)~=1
                            for farnaz=1:size(CH_not_controller,1)
                                if CH_cont_id(farnaz)==controller_t(zahra)

dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
                                    De2e(zahra,1)=De2e(zahra,1)+dis(farnaz)/(3*10^8);
                                end
                            end
                        end
                    elseif controller_t(zahra)==0
                        for farnaz=1:size(CH_not_controller,1)
                            if CH_cont_id(farnaz)==controller_t(zahra)
                                dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
                                De2e(zahra,1)=De2e(zahra,1)+dis(farnaz)/(3*10^8);
                            end
                        end
                    end
                end
            end
        end

        latency=De2e+1./(miu-lambda);
        for ttt=1:size(latency,1)
            if De2e(ttt,1)==0
                if lambda(ttt,1)==0
                    latency(ttt)=0;
                end
            end
        end

        %///////////////   BSA  ////////////////////////////////////////
        tic
        OL_C=[];
        LL_C=[];

        karim=1;
        asghar=1;
        for jafar=1:length(controller_t)
```

```
            if zahra~=length(controller_t)
                if node_locs(controller(jafar),8)~=1
                    if latency(jafar,1)==0
                        T_compare=0;
                    else
                        T_compare=latency(jafar,1)/Con_load(jafar,1);
                    end
                    if T_compare>T_treshold
                        OL_C(karim,1)=controller_t(jafar);
                        OL_C(karim,2)=latency(jafar,1);
                        OL_C(karim,3)=Con_load(jafar,1);
                        OL_C(karim,4)=jafar;
                        karim=karim+1;
                    else
                        LL_C(asghar,1)=controller_t(jafar);
                        LL_C(asghar,2)=latency(jafar,1);
                        LL_C(asghar,3)=Con_load(jafar,1);
                        LL_C(asghar,4)=jafar;
                        asghar=asghar+1;
                    end
                end
            elseif controller_t(zahra)==0
                if latency(jafar,1)==0
                    T_compare=0;
                else
                    T_compare=latency(jafar,1)/Con_load(jafar,1);
                end
                if T_compare>T_treshold
                    OL_C(karim,1)=controller_t(jafar);
                    OL_C(karim,2)=latency(jafar,1);
                    OL_C(karim,3)=Con_load(jafar,1);
                    OL_C(karim,4)=jafar;
                    karim=karim+1;
                else
                    LL_C(asghar,1)=controller_t(jafar);
                    LL_C(asghar,2)=latency(jafar,1);
                    LL_C(asghar,3)=Con_load(jafar,1);
                    LL_C(asghar,4)=jafar;
                    asghar=asghar+1;
                end
            end
        end

        %//////////////    BSW   ////////////////////////////////////////

        if ~isempty(OL_C)
            if ~isempty(LL_C)
                [OL_C_load_sort iiii]=sort(OL_C(:,3));
                for ee=1:size(OL_C,1)
                    cont_data=[];
                    Co=Con_load(OL_C(iiii(ee),4));
                    cont_data=cont_data_mat{OL_C(iiii(ee),4)};
                    [CH_load_sort uu]=sort(cont_data(:,3),'descend');
                    for rr=1:size(cont_data,1)
                        [LL_C_load_sort jjjj]=sort(LL_C(:,3));
                        for vv=1:size(LL_C,1)
                            CHo=cont_data(uu(rr),3);
                            if CHo+LL_C_load_sort(vv)<L_treshold
                                CH_cont_id(cont_data(uu(rr),1))=LL_C(jjjj(vv),1);
                                Con_load(OL_C(iiii(ee),4))=Con_load(OL_C(iiii(ee),4))-
CHo;

Con_load(LL_C(jjjj(vv),4))=Con_load(LL_C(jjjj(vv),4))+CHo;

                                break
```

```
                    end
                end
            end
        end
    end
end

%///////////  controller data update  ////////////////////////////

cont_data_mat={};
if ~isempty(CH_cont_id)
    for zzz=1:length(controller_t)
        cont_data=[];
        if ~isempty(CH_not_controller)
            for sss=1:size(CH_not_controller,1)
                if CH_cont_id(sss)==controller_t(zzz)
                    cont_data(1,1)=sss;
                    cont_data(1,2)=CH_not_controller(sss,1);
                    cont_data(1,3)=CH_load(sss,1);
                    cont_data(1,4)=controller_t(zzz);
                end
            end
        end
        cont_data_mat{zzz}=cont_data;
    end
end

%///////////  LFC  ////////////////////////////////////////////////
faulty_con=[];
if ~isempty(cont_fault_id)
    if rem(i,10)~=1 || length(cont_fault_id)==length(controller)
        for uu=1:length(controller_t)
            for vv=1:size(cont_fault_id,1)
                if cont_fault_id(vv,1)==controller_t(uu)
                    faulty_con(vv)=uu;
                end
            end
            for ww=1:size(cont_notfault_id,1)
                if cont_notfault_id(ww,1)==controller_t(uu)
                    notfaulty_con(ww)=uu;
                end
            end
        end

        LL_C_nf=[];

        asghar=1;
        for jafar=1:length(notfaulty_con)
            if Con_load(notfaulty_con(jafar),1)<L_treshold
                LL_C_nf(asghar,1)=controller_t(notfaulty_con(jafar));
                LL_C_nf(asghar,2)=Con_load(notfaulty_con(jafar),1);
                LL_C_nf(asghar,3)=notfaulty_con(jafar);
                asghar=asghar+1;
            end
        end

        if ~isempty(faulty_con)
            for ee=1:size(cont_fault_id,1)
                cont_data=[];
                cont_data=cont_data_mat{faulty_con(ee)};
                if ~isempty(cont_data)
                    [CH_load_sort uu]=sort(cont_data(:,3),'descend');
                    for rr=1:size(cont_data,1)
                        for vv=1:size(LL_C_nf,1)
                            CHo=cont_data(uu(rr),3);
```

```
                                    if CHo+LL_C_nf(vv,2)<L_treshold
                                        CH_cont_id(cont_data(uu(rr),1))=LL_C_nf(vv,1);

Con_load(LL_C_nf(vv,3))=Con_load(LL_C_nf(vv,3))+CHo;

                                        break
                                    end
                                end
                            end
                        end
                    end
                end
            end

        end
        BSAWF_time(i)=toc;
        controller_load_sort(1:length(controller),i)=Con_load(1:end-1,1);
        controller_load_sort(length(controller)+1,i)=Con_load(end,1);
        controller_id_sort(1:length(controller),i)=controller;

        controllers_load(controller,i)=Con_load(1:end-1,1);
        controllers_load(501,i)=Con_load(end,1);

        %////////////////  latency calculation update  ///////////////////////////

        lambda_2(:,i)=Con_load;
        Con_loc=[node_locs(controller,1:2);BS];
        CH_loc=node_locs(CH_not_controller,1:2);
        De2e_2=zeros(length(controller_t),1);

        if ~isempty(CH_cont_id)
            for zahra=1:length(controller_t)
                if ~isempty(CH_not_controller)
                    if zahra~=length(controller_t)
                        if node_locs(controller(zahra),8)~=1
                            for farnaz=1:size(CH_not_controller,1)
                                if CH_cont_id(farnaz)==controller_t(zahra)

dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));

De2e_2(zahra,1)=De2e_2(zahra,1)+dis(farnaz)/(3*10^8);
                                end
                            end
                        end
                    elseif controller_t(zahra)==0
                        for farnaz=1:size(CH_not_controller,1)
                            if CH_cont_id(farnaz)==controller_t(zahra)
                                dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
                                De2e_2(zahra,1)=De2e_2(zahra,1)+dis(farnaz)/(3*10^8);
                            end
                        end
                    end
                end
            end
        end

        latency_after_LFC(:,i)=De2e_2+1./(miu-lambda_2(:,i));

%         is_CH_fualty=zeros(CH_loc,1);
%         cost_1(i,T)=cost_SDN_mane(CH_loc,Con_loc(1:end-
1,:),is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node(),t_rate,EN,Con_node,con
_id,BS,w);

        %///////// communication phase //////////////
```

```
        for j=1:K
            curr_cluster=ch_members_mat{j};                % loop for communication
phase, copying cluster member to curr_cluster array
            curr_ch=cluster_heads(j);                      % copying cluster head for
current cluster

            if curr_ch~=1.5                                % if cluster head is not
NaN, go for communication phase
                for jj=1:size(curr_cluster)                % loop to send data for all
nodes
                    d_to_ch=sqrt( (node_locs(curr_cluster(jj),1) -
node_locs(curr_ch,1)).^2 + (node_locs(curr_cluster(jj),2) - node_locs(curr_ch,2)).^2);
% predicting distance of node to cluster head
                    Etx1=(L * Eelct) + (L * Efs * d_to_ch.^2);  % predicting transmit
energy for current node upto cluster head
                    if node_locs(curr_cluster(jj),5)>Etx1
                        node_locs(curr_cluster(jj),5) = node_locs(curr_cluster(jj),5) -
Etx1;   % deducting transmit energy for respective node
                        if which_clus_no_contr(j,1)==1
                            if T==1 || T==3
                                node_locs(curr_ch,5) = node_locs(curr_ch,5) - Erx(1);
% deduction receving energy for respective node
                            end
                        end
                    else
                        node_locs(curr_cluster(jj),5)=0;
                    end
                end

                if which_clus_no_contr(j,1)==1

                    if T==1 || T==3
                        ezat=find(CH_not_controller==curr_ch);
                        if ~isempty(CH_cont_id)
                            if CH_cont_id(ezat,1)==0

                                Etx2=(L * Eelct) + (L * Efs * d_to_BS(curr_ch).^2);
% deducting transmit energy upto BS from cluster head
                            else
                                id_cont=CH_cont_id(ezat,1);

d_to_cont=pdist2(node_locs(curr_ch,1:2),node_locs(id_cont,1:2));
                                Etx2=(L * Eelct) + (L * Efs * d_to_cont.^2);
                            end
                        end
                        if node_locs(curr_ch,5)>Etx2
                            TT=nodes_table{curr_ch};                    % using
alternative path using sdn concept
                            alt_node=TT(1,1);
                            dis=sqrt( (node_locs(alt_node,1) - node_locs(curr_ch,1)).^2
+ ...
                                (node_locs(alt_node,2) - node_locs(curr_ch,2)).^2);
% distance from nearest node (alternative path)
                            Etx2=(L * Eelct) + (L * Efs * dis.^2);
                            node_locs(curr_ch,5) = node_locs(curr_ch,5) - Etx2;
                        else
                            node_locs(curr_ch,5)=0;
                        end
                    else
                        TT=nodes_table{curr_ch};                    % using alternative
path using sdn concept
                        alt_node=TT(1,1);
                        dis=sqrt( (node_locs(alt_node,1) - node_locs(curr_ch,1)).^2 +
...
```

```
                            (node_locs(alt_node,2) - node_locs(curr_ch,2)).^2);       %
distance from nearest node (alternative path)
                        Etx2=(L * Eelct) + (L * Efs * dis.^2);
                        node_locs(curr_ch,5) = node_locs(curr_ch,5) - Etx2;
                    end
                end
            end
        end

        [r1,~]=find(node_locs(:,5) > 0);                    % checking the alive nodes
from its batteries

        if T==1 || T==3
            nodes_alive(i,1)=length(r1);                    % if without SDN save at
these locations
            residual_energy(i,1)=sum(node_locs(:,5));       % saving residual energy of
the ntw
        else
            nodes_alive_sdn(i,1)=length(r1);                % if with SDN save at these
locations
            residual_energy_sdn(i,1)=sum(node_locs(:,5));   % saving residual energy of
the ntw with SDN
        end

        if flag_2==0 && nodes_alive(i,1)<nodes
            flag_2=1;
            if T==1
                FND_GA=i;
            else
                FND_GRASP=i;
            end
        end

        %////////// ploting ////////////////////////////////////

        energy=node_locs(:,5);
        energy(controller)=[];
        if max(energy)==0                       % if all cluster heads over, break the
rounds loop
            if T==1
                LND_GA=i;
            else
                LND_GRASP=i;
            end
            break
        end

            if T==1
                figure(3)                                   % if without SDN make
the plot with figure 3
            else
                figure(4)
            end
            set(gca,'FontSize',20)
            set(gca, 'FontName', 'Times new roman')
            for ii=1:K                                      % ploting cluster nodes
with respective colors and markers
                [r,~]=find(idx==ii);
                plot(node_locs(r,1),node_locs(r,2),'s','color',colors(ii,:))
                hold on
            end
            plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20)
% ploting base station in same figure
            hold on
```

```
plot(node_locs(CH_not_controller,1),node_locs(CH_not_controller,2),'or','MarkerSize',10
,'MarkerFaceColor',[0 0 0]); %ploting cluster heads with black markers

plot(node_locs(controller,1),node_locs(controller,2),'ob','MarkerSize',15,'MarkerFaceCo
lor',[0 0 0]);
            axis([0 area(1)+20 0 area(2)+20]);

            if T==1
                title(strcat('(GA) Alive nodes:[',num2str(nodes_alive(i)),']
Round:[',num2str(i),']'))
                drawnow
            else
                title(strcat('(GRASP) Alive nodes:[',num2str(nodes_alive(i)),']
Round:[',num2str(i),']'))
                drawnow
            end
            hold off

        %%%%%%%%%% deploying faulty nodes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


fault_probability(i)=ceil(fault_percentage(ceil(i/(t_max/length(fault_percentage))))*no
des);
        faulty_nodes=randi([1,nodes],fault_probability(i),1);
        node_locs(faulty_nodes,8)=1;
        node_locs(faulty_nodes,5)=0;
        cont_fault_id=controller(find(node_locs(controller,8)==1));       % id of
faulty controllers
        cont_notfault_id=controller(find(node_locs(controller,8)==0));
        cont_notfault_id=[cont_notfault_id 0];
        flag_cont=0;
        cont_delete=[];

        if rem(i,10)==0 ||  length(cont_fault_id)==length(controller)
            for esi=1:length(controller)
                for iii=1:length(cont_fault_id)
                    if cont_fault_id(iii)==controller(esi)
                        cont_delete=[cont_delete esi];
                    end
                end
            end
%             controller(cont_delete)=[];
            flag_cont=length(cont_delete);
        end
    end

    %//////////////  plot  ////////////////////////////////////////////////
    % controllers load vs time
    sum_con_load=sum(controllers_load,2);
    controller_load_real=controllers_load(~(sum_con_load==0),:);

    if T==1
        figure(5)
    else
        figure(6)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')

    for dedede=1:size(controller_load_sort,1)
        if dedede==size(controller_load_sort,1)
        plot(controller_load_sort(dedede,:),'DisplayName',strcat('BS'));
        hold on
        else
```

```
            plot(controller_load_sort(dedede,:),'DisplayName',strcat('cont
['num2str(dedede),'] '));
            hold on
        end
    end
    legend
    xlabel('time(sec)')
    ylabel('controllers load')

    for david=2:i
        for john=1:n_cont_des
            if controller_id_sort(john,david)-controller_id_sort(john,david-1)

p=plot(david,controller_load_sort(john,david),'or','MarkerSize',10,'MarkerFaceColor',[0
0 0]);

set(get(get(p,'Annotation'),'LegendInformation'),'IconDisplayStyle','off');
            end
        end
    end

%     axis([0 250 0 300]);

    if T==1
        title(strcat('controllers load vs time for ['num2str(n_cont_des),'] alive
controllers and BS for GA'))
    else
        title(strcat('controllers load vs time for ['num2str(n_cont_des),'] alive
controllers and BS for GRASP'))
    end
    xlabel('time(sec)')
    ylabel('controllers load')

    % controllers load vs responce time
    if T==1
        figure(7)
    else
        figure(8)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')
    for dedede=1:size(lambda_2,1)
        if dedede==size(lambda_2,1)

plot(lambda_2(dedede,:),latency_after_LFC(dedede,:)*1000,'DisplayName',strcat('BS'));
            hold on
        else

plot(lambda_2(dedede,:),latency_after_LFC(dedede,:)*1000,'DisplayName',strcat('cont
['num2str(dedede),'] '));
            hold on
        end
    end
    if T==1
        title(strcat('latency vs controller load for all controllers in GA'))
    else
        title(strcat('latency vs controller load for all controllers in GRASP'))
    end

    legend
    xlabel('controller load')
    ylabel('latency(ms)')

    % mean response time of all controllers vs flow rate
    if T==1
```

```
        figure(14)
    else
        figure(15)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')
    avg_latency=sum(latency_after_LFC)./(n_cont_des+1);
    plot(100:100:ceil(i/10)*100,avg_latency(1:10:i)*1000);


    if T==1
        title(strcat('mean response time of all controllers vs flow rate in GA'))
    else
        title(strcat('mean response time of all controllers vs flow rate in GRASP'))
    end

    xlabel('flow rate')
    ylabel('response time(ms)')

    % controllers avg response time vs flow rate
    if T==1
        figure(16)
    else
        figure(17)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')

    for dedede=1:size(lambda_2,1)
        if dedede==size(lambda_2,1)

plot(100:100:ceil(i/10)*100,latency_after_LFC(dedede,1:10:i)*1000,'DisplayName',strcat(
'BS'));
            hold on
        else

plot(100:100:ceil(i/10)*100,latency_after_LFC(dedede,1:10:i)*1000,'DisplayName',strcat(
'cont [',num2str(dedede),'] '));
            hold on
        end
    end

    if T==1
        title(strcat('controllers avg response time vs flow rate in GA'))
    else
        title(strcat('controllers avg response time vs flow rate in GRASP'))
    end

    legend
    xlabel('flow rate')
    ylabel('response time(ms)')

    % percent of successful packets received vs time
    if T==1
        figure(12)
    else
        figure(13)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')

    plot(1:i,nodes_alive(:,1)*100/nodes);
    if T==1
        title(strcat('percent of successful packets received vs time in GA'))
    else
```

```
        title(strcat('percent of successful packets received vs time in GRASP'))
    end
    xlabel('time(s)')
    ylabel('percent of successful packets')


    if T==1
        disp(strcat('the total latency for GA is
[',num2str(sum(sum(latency_after_LFC))),']'))
    else
        disp(strcat('the total latency for GRASP is
[',num2str(sum(sum(latency_after_LFC))),']'))
    end


    % total time it takes to execute BSA/BSW/LFC them vs time
    if T==1
        figure(18)
    else
        figure(19)
    end
    set(gca,'FontSize',20)
    set(gca, 'FontName', 'Times new roman')

    plot(BSAWF_time);
    if T==1
        title(strcat('total time it takes to execute BSA/BSW/LFC vs time in GA'))
    else
        title(strcat('total time it takes to execute BSA/BSW/LFC vs time in GRASP'))
    end
    xlabel('time(s)')
    ylabel('time it takes to execute')
end
disp(strcat('w1 is [',num2str(w(1)),'] w2 is [',num2str(w(2)),'] w3 is
[',num2str(w(3)),'] w4 is [',num2str(w(4)),']'))
disp(strcat('the optimal cost for GA is [',num2str(optimalcost(1)),'] at iteration
[',num2str(iteration_optimal),'] and for GRASP is [',num2str(optimalcost(3)),']'))
disp(strcat('time it takes to reach optimal solution is [',num2str(time_opt_t(1)),']
for GA and [',num2str(time_opt_t(3)),'] for GRASP'))
if isempty(LND_GA)
    disp('some nodes are still alive in GA')
else
    disp(strcat('LND for GA is [',num2str(LND_GA),']'))
end

if isempty(LND_GRASP)
    disp('some nodes are still alive in GRASP')
else
    disp(strcat('LND for GRASP is [',num2str(LND_GRASP),']'))
end

if isempty(FND_GA)
    disp('all nodes are alive in GA')
else
    disp(strcat('FND for GA is [',num2str(FND_GA),']'))
end

if isempty(FND_GRASP)
    disp('all nodes are alive in GRASP')
else
    disp(strcat('FND for GRASP is [',num2str(FND_GRASP),']'))
end


% controllers load vs time for one controller
```

```
time=1:300;
t_rate=(ceil(time/10)*100)/nodes;
lambda=nodes*t_rate;

dis=pdist2(node_locs(:,1:2),BS);

res_time=sum(dis)/(300*10^6)+1./(miu-lambda);

figure(9)
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
plot(lambda,res_time*1000)
Y = res_time(1:10:end)'*1000;
title(strcat('latency vs controller load for one controller'))
xlabel('controller load')
ylabel('latency(ms)')
beep
```

Appendix C2: The code of the ALBATROSS Scheme

```
clear all
close all
clc
%%%%%%%%%%     Optimization parameters
n_cont_des=input('please enter number of desired controllers\n');                  %
initializing number of desired controllers
EN=2;                           % initializing energy consumed for transmission
%%%%%%%%%%
%/////////////// user parameters ///////////////////
nodes=500;                      % initializing number of nodes
area=[200 200];                 % initializing area of deployment
BS=area/2;                      % initializing base station location
Efs=10e-12;                     % initializing comunication constant values as per base paper
Ems=0.0013e-12;                 % initializing comunication constant values as per base paper
Eelct=50e-9;                    % initializing comunication constant values as per base paper
L0=4000;
L=L0;                           % initializing comunication constant values as per base paper
(packets)
E0=0.5;                         % initializing comunication constant values as per base paper
(initial energy)
Eda=5e-9;                       % initializing comunication constant values as per base paper
(data aggregation)
d0=88;                          % initializing comunication constant values as per base paper
(distance threshold)
Eo=0.1;
% fault_percentage=[0.01;0.05;0.1;0.2;0.3;0.4;0.5;0.6;0.7];
fault_percentage=[0.05e-2];
t_max=300;
T_treshold=2.63e-3;
miu=3000;
L_treshold=2600;
w=[25/48,13/48,7/48,3/48];
FND_GA=[];
LND_GA=[];
FND_GRASP=[];
LND_GRASP=[];
notations={'sb','sc','sr','sg','sm','sy','sk','ob','oc','or','og','om','oy','ok','db','dc','d
r','dg','dm','dy','dk','*b','*c','*r','*g','*m','*y','*k'};                          %
clusters color and marker style
%/////////// deplying nodes ////////////////////
hl_flag=0;
for T=1:2:3 % T=1 without sdn, T=2 with SDN
nodes_alive=[];
flag_2=0;
node_locs=[]; % initializing node locations
while(size(node_locs,1)<nodes) % deployment till we deploy all nodes
for i=1:ceil(2*(area(1)*area(2)/nodes)^0.5):area(1)-1 % deployment grid wise row with 20x20
for j=1:ceil(2*(area(1)*area(2)/nodes)^0.5):area(2)-1 % deployment grid wise colm with 20x20
```

```
y=randi([i i+ceil(2*(area(1)*area(2)/nodes)^0.5)-1],1,1); % randomly taking node position X
x=randi([j j+ceil(2*(area(1)*area(2)/nodes)^0.5)-1],1,1); % randomly taking node position Y
node_locs=[node_locs;x y]; % saving node in node locations array
if(size(node_locs,1)==nodes) % break the deployment if we deploy all nodes
break
end
end
if(size(node_locs,1)==nodes) % break the outerloop if we deploy all nodes
break
end
end
end
%///////// predicting alternative paths for sdn ///
for ii=1:nodes
curr_node=node_locs(i,:); % saving alternative paths for every node in sdn
dis=sqrt( (node_locs(:,1)-curr_node(1)).^2 + (node_locs(:,2)-curr_node(2)).^2);
com=[(1:nodes)' dis];
com=sortrows(com,2);
nodes_table{ii}=com(2:end,:);
end
%////////////////////////////////////////////////
figure(1) % fiugre handler
plot(node_locs(:,1),node_locs(:,2),'ok','MarkerFaceColor',[1,0,0]) % plotting all nodes
hold on
plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20) % plotting base
station location
axis([0 area(1) 0 area(2)+60]); % making axis limits to show all nodes and base station
hold off
%////////////////////// Similarity Graph prediction /////////////////
X=node_locs; % copying node locations in X variable
Y=node_locs; % copying node locations in Y variable
E = pdist2(X,Y); % predicting elucidian distance of every node to other node
E = E / max(E(:)); % normalizing distances with in range [0 1]
sigma=0.3; % taking sigma as 0.3
for i=1:size(E,1) % calculating similarity matrix
for j=1:size(E,2)
if ~(i==j) % as per the equation (5) from base paper
W(i,j)=exp(-1*(E(i,j)^2/(2*sigma^2)));
else
W(i,j)=0;
end
end
end
Adj_mat=W; % copying W variable to adjacency matrix
Deg_mat(1:size(Adj_mat,1),1:size(Adj_mat,2))=0; % initilizing degree matrix
for i=1:size(Adj_mat,1) % copying row wise to calculate degree matrix as per the base paper
Deg_mat(i,i)=sum(Adj_mat(:,i));
end
Lap_mat=Deg_mat - Adj_mat; % calculating laplacian matrix as per the base paper L=D-A
for i=1:length(Lap_mat)
```

```
normalised_Lap_mat(:,i) = Lap_mat(:,i) / Lap_mat(i,i); % making laplacian matrix to
normalized laplacian
end
d_to_BS=sqrt((node_locs(:,1) - BS(1)).^2 + (node_locs(:,2) - BS(2)).^2); % predicting
distances of all nodes to Base station
d_to_BS_avg=(mean(d_to_BS)^2); % taking average of distances
K=round((sqrt(nodes)/sqrt(2*pi)) * sqrt(Efs/Ems) * (sqrt(area(1)*area(2))/(d_to_BS_avg))); %
predicting K as per base paper
%/////////// clustring using k means ////////////////////////////////
for i=1:length(d_to_BS) % calculating D as per the base paper
D(i,i)=d_to_BS(i);
end
A=(D*-1/2) * Adj_mat * (D*-1/2); % A'=D1/2 A D-1/2
[VL,D1] = eig(A'); % calculating eigen values VL
figure(2)
idx = kmeans(VL(:,1:K),K);
colors=distinguishable_colors(K);
for i=1:K
[r,c]=find(idx==i); % loop to check which node belongs to which cluster decided by k means
plot(node_locs(r,1),node_locs(r,2),'s','color',colors(i,:)) % ploting the respective nodes
with predefined color and marker style
hold on
end
plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20) % plotting base
station location
axis([0 area(1) 0 area(2)+60]);
hold off
title('Clustering results of the KSCA-WSN algorithm')
drawnow
%////////// communication phase ////////////////////////////////////
node_locs(:,3)=d_to_BS; % saving base station distances at 3rd colm in node locations array
node_locs(:,4)=idx; % saving cluster number at 4th colm in node locations array
node_locs(:,5)=E0; % saving initial energies at 5th colm in node locations array
node_locs(:,6)=1:nodes; % saving id number at 6th colm in node locations array
node_locs(:,8)=0;
for i=1:size(node_locs,1) % loop to predict transmit energies for all nodes to base station
if(d_to_BS(i)<d0)
Etx(i,1)=(L * Eelct) + (L * Efs * d_to_BS(i).^2); % predicing as per equation 2 in base paper
else
Etx(i,1)=(L * Eelct) + (L * Efs * d_to_BS(i).^4);
end
end
controller=[]; % id of controllers
which_clus_no_contr=ones(K,1);
cont_fault_flag=0;
cont_fault_id=[];
controllers_load=zeros(nodes+1,t_max);
flag_cont=0;
BSAWF_time=[];
for i=1:t_max % loop to run for 300 secs
```

```
t_rate=(ceil(i/10)*100)/nodes;
L=L0*t_rate;
Erx(1:nodes,1)=L * Eelct; % predicing reciving energy for all nodes
Eagg(1:nodes,1)=L * Eda; % predicing data aggregation energy for all nodes
%//////// cluster head selection /////////////////
hl_flag=~hl_flag;
for j=1:K % loop to predict cluster heads for all clusters
[r,~]=find(node_locs(:,4)==j); % predicing nodes in jth cluster
cluster_nodes=node_locs(r,:); % copying respective nodes in cluster_nodes array
Sk=size(cluster_nodes,1); % number of nodes belongs to jth cluster Sk
if which_clus_no_contr(j)==1
for jj=1:size(cluster_nodes,1) % loop to preict Ermin as per base paper equation (7)
Ermin(jj,1) = abs(Sk) * ((Erx(r(jj)) + Eagg(r(jj)))) + Eo;
if(cluster_nodes(jj,5) > Ermin(jj)) % in respective node is above the Emin make the node as
cluster head selection process
cluster_nodes(jj,7)=1; % making 1 will make the node to enter in cluster head selection
process
else
cluster_nodes(jj,7)=0; % 0 will make the node not to enter in CH selection process
end
end
[r1,~]=find(cluster_nodes(:,7)==1); % checking the nodes which qualifies CH selection process
if ~(isempty(r1))
ch_members=cluster_nodes(r1,6);
DD=60; % defining the degree as 60
curr_energies=cluster_nodes(r1,:); % reading cluster member energies
curr_energies=sortrows(curr_energies,5);
[rl,~]=find(curr_energies(:,5)>Eo & curr_energies(:,5)<E0/2);
[rh,~]=find(curr_energies(:,5)>=E0/2 & curr_energies(:,5)<=E0);
if(~isempty(rl))
rl=1;
end
if(~isempty(rh))
rh=1;
end
HE_y=mean(curr_energies(rh,1)); % taking mean of y position of higher E nodes
HE_x=mean(curr_energies(rh,2)); % taking mean of x position of higher E nodes
LE_y=mean(curr_energies(rl,1)); % taking mean of y position of lesser E nodes
LE_x=mean(curr_energies(rl,2)); % taking mean of x position of lesser E nodes
X=cluster_nodes(r1,1); % checking x positions of all cluster nodes
Y=cluster_nodes(r1,2); % checking y positions of all cluster nodes
dis_cal_h=sqrt( (X-HE_y).^2 + (Y-HE_x).^2); % calculating distances from High energy nodes
center
[rh,ch]=find(dis_cal_h==min(dis_cal_h)); % calculating distances from High energy nodes
center
xx=cosd(DD)*(X - X(1)) - sin(DD)*(Y-Y(1)) + X(1); % checking DD degree from center
yy=cosd(DD)*(X - X(1)) - sin(DD)*(Y-Y(1)) + X(1); % checking DD degree from center
dis_cal_h=sqrt( (X-xx).^2 + (Y-yy).^2); % checking nearest node at current degree
[rh1,ch1]=find(dis_cal_h==min(dis_cal_h)); % checking node id
ch_Hid=rh1(1); % copying next cluster head id
```

```
dis_cal_l=sqrt( (X-LE_y).^2 + (Y-LE_x).^2); % calculating distances from High energy nodes
center
[rl,cl]=find(dis_cal_l==min(dis_cal_l)); % calculating distances from High energy nodes
center
xx=cosd(DD)*(X - X(1)) - sin(DD)*(Y-Y(1)) + X(1); % checking DD degree from center
yy=cosd(DD)*(X - X(1)) - sin(DD)*(Y-Y(1)) + X(1); % checking DD degree from center
dis_cal_l=sqrt( (X-xx).^2 + (Y-yy).^2); % checking nearest node at current degree
[rl1,cl1]=find(dis_cal_l==min(dis_cal_l)); % checking node id
ch_Lid=rl1(1);
if(hl_flag==0) % toggling cluster id one time from high area another time from low
cluster_heads(j,1)=ch_Hid;
else
cluster_heads(j,1)=ch_Lid;
end
cluster_heads(j,1)=ch_members(randi([1 size(ch_members,1)],1,1));
cluster_nodes_t=cluster_nodes;
cluster_nodes_t(find(cluster_nodes_t(:,6)==cluster_heads(j,1)),:)=[];
simin=find(cluster_nodes_t(:,5)==0);
cluster_nodes_t(simin,:)=[];
ch_members_mat{j}=cluster_nodes_t(:,6);
else
ch_members_mat{j}=[]; % otherwise if no CH node qualifies, make the CH for that cluster to
NaN
cluster_heads(j,1)=1.5; % a flag to show that there is no CH in the cluster
node_locs(cluster_nodes(:,6),5)=0;
end
end
cluster_nodes_t_id=ch_members_mat{j};
cluster_nodes_tt=node_locs(cluster_nodes_t_id,:);
simin=find(cluster_nodes_tt(:,5)==0);
cluster_nodes_tt(simin,:)=[];
ch_members_mat{j}=cluster_nodes_tt(:,6);
end
remain_CH=cluster_heads;
remain_CH(find(remain_CH==1.5))=[];
%%%%%%%%%%%%%%%%% FFFFFFFFFFFFFFAAAAAARRRRRRRRRRRRnaz controller selection
n_cont=n_cont_des-length(controller)+flag_cont;
for javad=1:K
CH_node(javad,1)=length(ch_members_mat{javad});
end
%////// evaluating number of alive nodes in each cluster ////////
c_faulty_node=zeros(K,1);
for ezat=1:K
c_members=ch_members_mat{ezat};
for rajab=1:size(c_members,1)
if node_locs(c_members(rajab),8)==1
c_faulty_node(ezat,1)=c_faulty_node(ezat,1)+1;
end
end
CH_alive_node(ezat,1)=CH_node(ezat,1)-c_faulty_node(ezat,1);
```

```
end
%////////////////////////////////////////////////////////////////
CH_not_controller=remain_CH;
for naghme=1:length(controller)
CH_not_controller(find(CH_not_controller==controller(naghme)))=[];
end
is_CH_fualty=zeros(length(CH_not_controller),1);
if n_cont>0
if size(CH_not_controller,1)>n_cont
if T==1
tic
[cont_opt,CHrem,cost_opt,iteration,for_plot,meann]=GA_SDN(n_cont,CH_not_controller,node_locs(
CH_not_controller,1:2),is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node,t_rate,EN,BS
,w);
time_opt(T)=toc;
elseif T==3
tic
[cont_opt,CHrem,cost_opt]=GRASP_SDN(n_cont,CH_not_controller,node_locs(CH_not_controller,1:2)
,is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node,t_rate,EN,BS,cluster_heads,w);
time_opt(T)=toc;
end
else
cont_opt=CH_not_controller';
end
if i==1
controller=cont_opt;
else
ema=1;
for suzan=1:length(cont_opt)
controller(cont_delete(suzan))=cont_opt(suzan);
end
end
cont_opt=[];
end
if i==1
time_opt_t=time_opt;
if T==1
optimalcost(T)=cost_opt;
iteration_optimal=iteration;
figure(10)
plot(for_plot)
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
title(strcat('minimum cost value vs iteration for GA'))
xlabel('Iteration')
ylabel('minimum cost')
figure(11)
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
plot(meann)
```

```
title(strcat('average cost value vs iteration for GA'))
xlabel('Iteration')
ylabel('average cost')
elseif T==3
optimalcost(T)=cost_opt;
end
end
%%%% clustring CH among controllers
if ~isempty(controller)
for asghar=1:n_cont_des
CH_not_controller(find(CH_not_controller==controller(asghar)))=[];
end
end
which_clus_no_contr=ones(K,1);
which_clus_no_contr(node_locs(controller,4),1)=0;
if ~isempty(CH_not_controller)
CH_cont_id=CH_cont_id_fun(node_locs(CH_not_controller,1:2),[node_locs(controller,1:2);BS],[co
ntroller';0]);
end
%////////////////////// load calculation ////////////////////
CH_load=t_rate*CH_alive_node;
Con_load=[];
for ezat=1:length(controller)
Con_load(ezat,1)=CH_load(find(cluster_heads==controller(ezat))); % initializing controller
load matrix
end
Con_load(length(controller)+1,1)=0;
controller_t=[controller 0];
if ~isempty(CH_cont_id)
for zzz=1:length(controller_t)
cont_data=[];
if ~isempty(CH_not_controller)
for sss=1:size(CH_not_controller,1)
if CH_cont_id(sss)==controller_t(zzz)
Con_load(zzz,1)=Con_load(zzz,1)+CH_load(sss,1); % calculating controllers load
cont_data(1,1)=sss;
cont_data(1,2)=CH_not_controller(sss,1);
cont_data(1,3)=CH_load(sss,1);
cont_data(1,4)=controller_t(zzz);
end
end
end
cont_data_mat{zzz}=cont_data;
end
end
%///////////////// latency calculation //////////////////////////
lambda=Con_load;
Con_loc=[node_locs(controller,1:2);BS];
CH_loc=node_locs(CH_not_controller,1:2);
De2e=zeros(length(controller_t),1);
```

```
if ~isempty(CH_cont_id)
for zahra=1:length(controller_t)
if ~isempty(CH_not_controller)
if zahra~=length(controller_t)
if node_locs(controller(zahra),8)~=1
for farnaz=1:size(CH_not_controller,1)
if CH_cont_id(farnaz)==controller_t(zahra)
dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
De2e(zahra,1)=De2e(zahra,1)+dis(farnaz)/(3*10^8);
end
end
end
elseif controller_t(zahra)==0
for farnaz=1:size(CH_not_controller,1)
if CH_cont_id(farnaz)==controller_t(zahra)
dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
De2e(zahra,1)=De2e(zahra,1)+dis(farnaz)/(3*10^8);
end
end
end
end
end
end
latency=De2e+1./(miu-lambda);
for ttt=1:size(latency,1)
if De2e(ttt,1)==0
if lambda(ttt,1)==0
latency(ttt)=0;
end
end
end
%/////////////// BSA ////////////////////////////////////////
tic
OL_C=[];
LL_C=[];
karim=1;
asghar=1;
for jafar=1:length(controller_t)
if zahra~=length(controller_t)
if node_locs(controller(jafar),8)~=1
if latency(jafar,1)==0
T_compare=0;
else
T_compare=latency(jafar,1)/Con_load(jafar,1);
end
if T_compare>T_treshold
OL_C(karim,1)=controller_t(jafar);
OL_C(karim,2)=latency(jafar,1);
OL_C(karim,3)=Con_load(jafar,1);
OL_C(karim,4)=jafar;
```

```
karim=karim+1;
else
LL_C(asghar,1)=controller_t(jafar);
LL_C(asghar,2)=latency(jafar,1);
LL_C(asghar,3)=Con_load(jafar,1);
LL_C(asghar,4)=jafar;
asghar=asghar+1;
end
end
elseif controller_t(zahra)==0
if latency(jafar,1)==0
T_compare=0;
else
T_compare=latency(jafar,1)/Con_load(jafar,1);
end
if T_compare>T_treshold
OL_C(karim,1)=controller_t(jafar);
OL_C(karim,2)=latency(jafar,1);
OL_C(karim,3)=Con_load(jafar,1);
OL_C(karim,4)=jafar;
karim=karim+1;
else
LL_C(asghar,1)=controller_t(jafar);
LL_C(asghar,2)=latency(jafar,1);
LL_C(asghar,3)=Con_load(jafar,1);
LL_C(asghar,4)=jafar;
asghar=asghar+1;
end
end
end
%/////////////// BSW ////////////////////////////////////////
if ~isempty(OL_C)
if ~isempty(LL_C)
[OL_C_load_sort iiii]=sort(OL_C(:,3));
for ee=1:size(OL_C,1)
cont_data=[];
Co=Con_load(OL_C(iiii(ee),4));
cont_data=cont_data_mat{OL_C(iiii(ee),4)};
[CH_load_sort uu]=sort(cont_data(:,3),'descend');
for rr=1:size(cont_data,1)
[LL_C_load_sort jjjj]=sort(LL_C(:,3));
for vv=1:size(LL_C,1)
CHo=cont_data(uu(rr),3);
if CHo+LL_C_load_sort(vv)<L_treshold
CH_cont_id(cont_data(uu(rr),1))=LL_C(jjjj(vv),1);
Con_load(OL_C(iiii(ee),4))=Con_load(OL_C(iiii(ee),4))-CHo;
Con_load(LL_C(jjjj(vv),4))=Con_load(LL_C(jjjj(vv),4))+CHo;
break
end
end
```

```
end
end
end
end
%////////// controller data update ////////////////////////////
cont_data_mat={};
if ~isempty(CH_cont_id)
for zzz=1:length(controller_t)
cont_data=[];
if ~isempty(CH_not_controller)
for sss=1:size(CH_not_controller,1)
if CH_cont_id(sss)==controller_t(zzz)
cont_data(1,1)=sss;
cont_data(1,2)=CH_not_controller(sss,1);
cont_data(1,3)=CH_load(sss,1);
cont_data(1,4)=controller_t(zzz);
end
end
end
cont_data_mat{zzz}=cont_data;
end
end
%////////// LFC ////////////////////////////////////////////
faulty_con=[];
if ~isempty(cont_fault_id)
if rem(i,10)~=1 || length(cont_fault_id)==length(controller)
for uu=1:length(controller_t)
for vv=1:size(cont_fault_id,1)
if cont_fault_id(vv,1)==controller_t(uu)
faulty_con(vv)=uu;
end
end
for ww=1:size(cont_notfault_id,1)
if cont_notfault_id(ww,1)==controller_t(uu)
notfaulty_con(ww)=uu;
end
end
end
LL_C_nf=[];
asghar=1;
for jafar=1:length(notfaulty_con)
if Con_load(notfaulty_con(jafar),1)<L_treshold
LL_C_nf(asghar,1)=controller_t(notfaulty_con(jafar));
LL_C_nf(asghar,2)=Con_load(notfaulty_con(jafar),1);
LL_C_nf(asghar,3)=notfaulty_con(jafar);
asghar=asghar+1;
end
end
if ~isempty(faulty_con)
for ee=1:size(cont_fault_id,1)
```

```
cont_data=[];
cont_data=cont_data_mat{faulty_con(ee)};
if ~isempty(cont_data)
[CH_load_sort uu]=sort(cont_data(:,3),'descend');
for rr=1:size(cont_data,1)
for vv=1:size(LL_C_nf,1)
CHo=cont_data(uu(rr),3);
if CHo+LL_C_nf(vv,2)<L_treshold
CH_cont_id(cont_data(uu(rr),1))=LL_C_nf(vv,1);
Con_load(LL_C_nf(vv,3))=Con_load(LL_C_nf(vv,3))+CHo;
break
end
end
end
end
end
end
end
end
BSAWF_time(i)=toc;
controller_load_sort(1:length(controller),i)=Con_load(1:end-1,1);
controller_load_sort(length(controller)+1,i)=Con_load(end,1);
controller_id_sort(1:length(controller),i)=controller;
controllers_load(controller,i)=Con_load(1:end-1,1);
controllers_load(501,i)=Con_load(end,1);
%///////////////// latency calculation update //////////////////////////
lambda_2(:,i)=Con_load;
Con_loc=[node_locs(controller,1:2);BS];
CH_loc=node_locs(CH_not_controller,1:2);
De2e_2=zeros(length(controller_t),1);
if ~isempty(CH_cont_id)
for zahra=1:length(controller_t)
if ~isempty(CH_not_controller)
if zahra~=length(controller_t)
if node_locs(controller(zahra),8)~=1
for farnaz=1:size(CH_not_controller,1)
if CH_cont_id(farnaz)==controller_t(zahra)
dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
De2e_2(zahra,1)=De2e_2(zahra,1)+dis(farnaz)/(3*10^8);
end
end
end
elseif controller_t(zahra)==0
for farnaz=1:size(CH_not_controller,1)
if CH_cont_id(farnaz)==controller_t(zahra)
dis(farnaz)=pdist2(CH_loc(farnaz,:),Con_loc(zahra,:));
De2e_2(zahra,1)=De2e_2(zahra,1)+dis(farnaz)/(3*10^8);
end
end
end
```

```
end
end
end
latency_after_LFC(:,i)=De2e_2+1./(miu-lambda_2(:,i));
% is_CH_fualty=zeros(CH_loc,1);
% cost_1(i,T)=cost_SDN_mane(CH_loc,Con_loc(1:end-
1,:),is_CH_fualty,node_locs(CH_not_controller,5),CH_alive_node(),t_rate,EN,Con_node,con_id,BS
,w);
%/////// communication phase /////////////
for j=1:K
curr_cluster=ch_members_mat{j}; % loop for communication phase, copying cluster member to
curr_cluster array
curr_ch=cluster_heads(j); % copying cluster head for current cluster
if curr_ch~=1.5 % if cluster head is not NaN, go for communication phase
for jj=1:size(curr_cluster) % loop to send data for all nodes
d_to_ch=sqrt( (node_locs(curr_cluster(jj),1) - node_locs(curr_ch,1)).^2 +
(node_locs(curr_cluster(jj),2) - node_locs(curr_ch,2)).^2); % predicting distance of node to
cluster head
Etx1=(L * Eelct) + (L * Efs * d_to_ch.^2); % predicting transmit energy for current node upto
cluster head
if node_locs(curr_cluster(jj),5)>Etx1
node_locs(curr_cluster(jj),5) = node_locs(curr_cluster(jj),5) - Etx1; % deducting transmit
energy for respective node
if which_clus_no_contr(j,1)==1
if T==1 || T==3
node_locs(curr_ch,5) = node_locs(curr_ch,5) - Erx(1); % deduction receving energy for
respective node
end
end
else
node_locs(curr_cluster(jj),5)=0;
end
end
if which_clus_no_contr(j,1)==1
if T==1 || T==3
ezat=find(CH_not_controller==curr_ch);
if ~isempty(CH_cont_id)
if CH_cont_id(ezat,1)==0
Etx2=(L * Eelct) + (L * Efs * d_to_BS(curr_ch).^2); % deducting transmit energy upto BS from
cluster head
else
id_cont=CH_cont_id(ezat,1);
d_to_cont=pdist2(node_locs(curr_ch,1:2),node_locs(id_cont,1:2));
Etx2=(L * Eelct) + (L * Efs * d_to_cont.^2);
end
end
if node_locs(curr_ch,5)>Etx2
TT=nodes_table{curr_ch}; % using alternative path using sdn concept
alt_node=TT(1,1);
dis=sqrt( (node_locs(alt_node,1) - node_locs(curr_ch,1)).^2 + ...
```

```
(node_locs(alt_node,2) - node_locs(curr_ch,2)).^2); % distance from nearest node (alternative
path)
Etx2=((L * Eelct) + (L * Efs * dis.^2))/8;
node_locs(curr_ch,5) = node_locs(curr_ch,5) - Etx2;
else
node_locs(curr_ch,5)=0;
end
else
TT=nodes_table{curr_ch}; % using alternative path using sdn concept
alt_node=TT(1,1);
dis=sqrt( (node_locs(alt_node,1) - node_locs(curr_ch,1)).^2 + ...
(node_locs(alt_node,2) - node_locs(curr_ch,2)).^2); % distance from nearest node (alternative
path)
Etx2=(L * Eelct) + (L * Efs * dis.^2);
node_locs(curr_ch,5) = node_locs(curr_ch,5) - Etx2;
end
end
end
end
[r1,~]=find(node_locs(:,5) > 0); % checking the alive nodes from its batteries
if T==1 || T==3
nodes_alive(i,1)=length(r1); % if with SDN save at these locations
residual_energy(i,1)=sum(node_locs(:,5)); % saving residual energy of the ntw with SDN
else
nodes_alive_sdn(i,1)=length(r1); % if with SDN save at these locations
residual_energy_sdn(i,1)=sum(node_locs(:,5)); % saving residual energy of the ntw with SDN
end
if flag_2==0 && nodes_alive(i,1)<nodes
flag_2=1;
if T==1
FND_GA=i;
else
FND_GRASP=i;
end
end
%////////// ploting ////////////////////////////////////
energy=node_locs(:,5);
energy(controller)=[];
if max(energy)==0 % if all cluster heads over, break the rounds loop
if T==1
LND_GA=i;
else
LND_GRASP=i;
end
break
end
if T==1
figure(3) % if without SDN make the plot with figure 3
else
figure(4)
```

```
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
for ii=1:K % ploting cluster nodes with respective colors and markers
[r,~]=find(idx==ii);
plot(node_locs(r,1),node_locs(r,2),'s','color',colors(ii,:))
hold on
end
plot(BS(1),BS(2),'sk','MarkerFaceColor',[153,217,234]/255,'MarkerSize',20) % ploting base
station in same figure
hold on
plot(node_locs(CH_not_controller,1),node_locs(CH_not_controller,2),'or','MarkerSize',10,'Mark
erFaceColor',[0 0 0]); %ploting cluster heads with black markers
plot(node_locs(controller,1),node_locs(controller,2),'ob','MarkerSize',15,'MarkerFaceColor',[
0 0 0]);
axis([0 area(1)+20 0 area(2)+20]);
if T==1
title(strcat('(GA) Alive nodes:[',num2str(nodes_alive(i)),'] Round:[',num2str(i),']'))
drawnow
else
title(strcat('(GRASP) Alive nodes:[',num2str(nodes_alive(i)),'] Round:[',num2str(i),']'))
drawnow
end
hold off
%%%%%%%%%% deploying faulty nodes %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fault_probability(i)=ceil(fault_percentage(ceil(i/(t_max/length(fault_percentage))))*nodes);
faulty_nodes=randi([1,nodes],fault_probability(i),1);
node_locs(faulty_nodes,8)=1;
node_locs(faulty_nodes,5)=0;
cont_fault_id=controller(find(node_locs(controller,8)==1)); % id of faulty controllers
cont_notfault_id=controller(find(node_locs(controller,8)==0));
cont_notfault_id=[cont_notfault_id 0];
flag_cont=0;
cont_delete=[];
if rem(i,10)==0 || length(cont_fault_id)==length(controller)
for esi=1:length(controller)
for iii=1:length(cont_fault_id)
if cont_fault_id(iii)==controller(esi)
cont_delete=[cont_delete esi];
end
end
end
% controller(cont_delete)=[];
flag_cont=length(cont_delete);
end
end
%/////////////// plot ////////////////////////////////////////////////
% controllers load vs time
sum_con_load=sum(controllers_load,2);
controller_load_real=controllers_load(~(sum_con_load==0),:);
```

```
if T==1
figure(5)
else
figure(6)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
for dedede=1:size(controller_load_sort,1)
if dedede==size(controller_load_sort,1)
plot(controller_load_sort(dedede,:),'DisplayName',strcat('BS'));
hold on
else
plot(controller_load_sort(dedede,:),'DisplayName',strcat('cont [',num2str(dedede),'] '));
hold on
end
end
legend
xlabel('time(sec)')
ylabel('controllers load')
for david=2:i
for john=1:n_cont_des
if controller_id_sort(john,david)-controller_id_sort(john,david-1)
p=plot(david,controller_load_sort(john,david),'or','MarkerSize',10,'MarkerFaceColor',[0 0
0]);
set(get(get(p,'Annotation'),'LegendInformation'),'IconDisplayStyle','off');
end
end
end
% axis([0 250 0 300]);
if T==1
title(strcat('controllers load vs time for [',num2str(n_cont_des),'] alive controllers and BS
for GA'))
else
title(strcat('controllers load vs time for [',num2str(n_cont_des),'] alive controllers and BS
for GRASP'))
end
xlabel('time(sec)')
ylabel('controllers load')
% controllers load vs responce time
if T==1
figure(7)
else
figure(8)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
for dedede=1:size(lambda_2,1)
if dedede==size(lambda_2,1)
plot(lambda_2(dedede,:),latency_after_LFC(dedede,:)*1000,'DisplayName',strcat('BS'));
hold on
```

```
else
plot(lambda_2(dedede,:),latency_after_LFC(dedede,:)*1000,'DisplayName',strcat('cont
[',num2str(dedede),'] '));
hold on
end
end
if T==1
title(strcat('latency vs controller load for all controllers in GA'))
else
title(strcat('latency vs controller load for all controllers in GRASP'))
end
legend
xlabel('controller load')
ylabel('latency(ms)')
% mean response time of all controllers vs flow rate
if T==1
figure(14)
else
figure(15)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
avg_latency=sum(latency_after_LFC)./(n_cont_des+1);
plot(100:100:ceil(i/10)*100,avg_latency(1:10:i)*1000);
if T==1
title(strcat('mean response time of all controllers vs flow rate in GA'))
else
title(strcat('mean response time of all controllers vs flow rate in GRASP'))
end
xlabel('flow rate')
ylabel('response time(ms)')
% controllers avg response time vs flow rate
if T==1
figure(16)
else
figure(17)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
for dedede=1:size(lambda_2,1)
if dedede==size(lambda_2,1)
plot(100:100:ceil(i/10)*100,latency_after_LFC(dedede,1:10:i)*1000,'DisplayName',strcat('BS'))
;
hold on
else
plot(100:100:ceil(i/10)*100,latency_after_LFC(dedede,1:10:i)*1000,'DisplayName',strcat('cont
[',num2str(dedede),'] '));
hold on
end
end
```

```
if T==1
title(strcat('controllers avg response time vs flow rate in GA'))
else
title(strcat('controllers avg response time vs flow rate in GRASP'))
end
legend
xlabel('flow rate')
ylabel('response time(ms)')
% percent of successful packets received vs time
if T==1
figure(12)
else
figure(13)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
plot(1:i,nodes_alive(:,1)*100/nodes);
if T==1
title(strcat('percent of successful packets received vs time in GA'))
else
title(strcat('percent of successful packets received vs time in GRASP'))
end
xlabel('time(s)')
ylabel('percent of successful packets')
if T==1
disp(strcat('the total latency for GA is [',num2str(sum(sum(latency_after_LFC))),']'))
else
disp(strcat('the total latency for GRASP is [',num2str(sum(sum(latency_after_LFC))),']'))
end
% total time it takes to execute BSA/BSW/LFC them vs time
if T==1
figure(18)
else
figure(19)
end
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
plot(BSAWF_time);
if T==1
title(strcat('total time it takes to execute BSA/BSW/LFC vs time in GA'))
else
title(strcat('total time it takes to execute BSA/BSW/LFC vs time in GRASP'))
end
xlabel('time(s)')
ylabel('time it takes to execute')
end
disp(strcat('w1 is [',num2str(w(1)),'] w2 is [',num2str(w(2)),'] w3 is [',num2str(w(3)),'] w4
is [',num2str(w(4)),']'))
disp(strcat('the optimal cost for GA is [',num2str(optimalcost(1)),'] at iteration
[',num2str(iteration_optimal),'] and for GRASP is [',num2str(optimalcost(3)),']'))
```

```
disp(strcat('time it takes to reach optimal solution is [',num2str(time_opt_t(1)),'] for GA
and [',num2str(time_opt_t(3)),'] for GRASP'))
if isempty(LND_GA)
disp('some nodes are still alive in GA')
else
disp(strcat('LND for GA is [',num2str(LND_GA),']'))
end
if isempty(LND_GRASP)
disp('some nodes are still alive in GRASP')
else
disp(strcat('LND for GRASP is [',num2str(LND_GRASP),']'))
end
if isempty(FND_GA)
disp('all nodes are alive in GA')
else
disp(strcat('FND for GA is [',num2str(FND_GA),']'))
end
if isempty(FND_GRASP)
disp('all nodes are alive in GRASP')
else
disp(strcat('FND for GRASP is [',num2str(FND_GRASP),']'))
end
% controllers load vs time for one controller
time=1:300;
t_rate=(ceil(time/10)*100)/nodes;
lambda=nodes*t_rate;
dis=pdist2(node_locs(:,1:2),BS);
res_time=sum(dis)/(300*10^6)+1./(miu-lambda);
figure(9)
set(gca,'FontSize',20)
set(gca, 'FontName', 'Times new roman')
plot(lambda,res_time*1000)
Y = res_time(1:10:end)'*1000;
title(strcat('latency vs controller load for one controller'))
xlabel('controller load')
ylabel('latency(ms)')
beep
```