A Comparative Analysis of Deadlock Avoidance and Prevention Algorithms for Resource Provisioning in Intelligent Autonomous Transport Systems over 6G Infrastructure

Emeka Ugwuanyi, *Member, IEEE,* Muddesar Iqbal, *Member, IEEE,* and Tasos Dagiuklas, *Member, IEEE,*

6G is the future of intelligent connectivity artefacts with Artificial Intelligence (AI) at its backbone. The Multi-Access Edge Computing (MEC) based 6G enabled infrastructure helps in achieving the required zero latency for autonomous Intelligent Transport Systems (ITS) with features of low power consumption, lower end-to-end latency, minimal processing/transmission overheads, higher throughput and reliability. MEC are prone to a deadlock due to the limited amount of available computational resources, resulting in fatal delays in Vehicle-to-Vehicle (V2V), Vehicles to Road Side Unit (RSU) and RSU to ITS communication. The unresolved deadlock may entail higher energy consumption that can adversely affect the Quality of Service (QoS) in terms of safety and reliability with potential threats of causing fatal accidents. Therefore, it is almost imperative to resolve the deadlocks from MEC to comply with the QoS parameters of MEC based autonomous vehicles. The asserted goals can be achieved by employing an intelligent and adaptive deadlock resolution strategy. In this paper, a deadlock-aware, and collaborative edge decision algorithm has been proposed for facilitating the seamless communication of autonomous vehicles over MEC. Additionally, deadlock avoidance and prevention schemes have been evaluated using Bankers resource request avoidance algorithm, wound wait algorithm and wait-die algorithms for resource provisioning in collaborative MEC. Furthermore, the effectiveness of deadlock avoidance and prevention algorithms in real-time scenarios has been analyzed in MEC systems. The metrics used for a comparative analysis in this research include Round-trip time, Queue wait-time and CPU utilization. The proposed algorithm shows promising results when compared with prevalent techniques.

Index Terms—**Intelligent Autonomous Transport Systems, Collaborative Edge, MEC, deadlock, real-time algorithms, Banker's algorithm**

I. INTRODUCTION

MULTI-ACCESS Mobile Edge Computing (MEC) is one of the 6G enabling technologies proposed to meet the advertised ultra-low latency standards. According to [1][2] MEC is a hub of access points that has storage and computational capacity for a wide range of consumer devices such as Blockchain, terrestrial networks, IoT devices and autonomous vehicles (AVs) [3]. Low latency is crucial for the emerging AV networks that require ubiquitous user connectivity and real-time computational offload response. To achieve Ultra-reliable Low Latency Communication (URLLC) in 6G networks, QoS parameters of reliability and availability exhibit higher precedence.

These factors play an important role in ensuring the smooth delivery of time-critical applications and have been widely accepted as key concerns of network service providers [4] when coupled with AVs [5], [6]. In the current research, it is assumed that the AV network requires optimal computational and storage resources [7],[8]. Hence, a large portion of their workload is offloaded. Due to the proximity of MEC to end-users, the workload of these devices are offloaded to the nearest MEC [9][10].

A substantial increase in such resource-constrained AVs, in turn, increases the number of devices sharing and competing for the limited resources provided by the MEC platform[11]. Due to the limited amount of resources available on the edge node, there is a need to effectively manage MEC and AV resources to prevent over-provisioning of resources and deadlock[6]. Deadlock may arise as a process requests for resources that are held by another waiting resource, thereby leading to a circular wait state[12]. Such an unplanned circular wait would increase the energy consumption during offloading. It has been proved in our prior research endeavours[13][10] [14]. Additionally, it has been proved that the MEC platform is prone to a deadlock due to limited resource constraints. Therefore, effective precautions need to be used to avoid over-provisioning of the edge node to AV clients. Such that low latency can be maintained and deadlock in the AV network is eradicated. Deadlock is an undesirable phenomenon in real-time systems that hosts time-critical applications which are sensitive to latency. In [12], deadlock is defined as an event that happens in a multi-programming environment where several processes compete for a finite number of resources. During this, a process enters a waiting state when it requests resources that are not available at the time of the request. If the waiting process is never able to change state, because the resources it has requested are held by another waiting process, then the system is said to be in a deadlock. *Deadlock* has been studied extensively [15] [16] [17] in various multi-programming environments which can be applied to AVs. However, there is limited research on deadlock management in real-time systems in the context of MEC for AVs.

According to [12], real-time algorithms are used when a rigid time requirement has been placed on the operation of a processor or the flow of data. Therefore, it is used as a control mechanism in dedicated applications. IoT systems are usually real-time driven because the data obtained by the IoT sensors must be analyzed at a given time for timely and accurate decisions to be made. Therefore, it is important to consider the effect of deadlock strategies on a real-time system.

E. Ugwuanyi is with the Division of Computer Science and Informatics, London South Bank University, London SE1 0AA, e-mail: (ugwuanye@lsbu.ac.uk).

M. Iqbal and T. Dagiuklas are with London South Bank University.

One of the most important aspects of offloading is choosing the best candidate to offload a given task. This decision is crucial as a suitable candidate needs to be selected to avoid missing the task deadline. This decision must be optimized considering the computational resources of the MEC and network constraints to avoid re-offloading, overprovisioning of resources and deadlock in AV networks. This paper addresses the highlighted problems by proposing a deadlock aware collaborative edge computing offloading algorithm to effectively select the best candidate for offload within an AV network. The proposed algorithm ensures reliability and low latency network communication. Furthermore, the current work aims to provide a comparative analysis of different strategies that can be used in building a real-time and reliable 6G network system by minimizing the chances of deadlock during resource provisioning for AVs/IoT devices in MEC. Hence, an evaluation and comparison of different deadlock avoidance and prevention algorithms have been made. In this regard, six algorithms have been examined using three deadlock strategies (*Bankers algorithm*, *wound wait*, and *wait die*) and 2 real-time schemes (Earliest Deadline First and Rate Monotonic Scheduling). These algorithms were examined due to their effectiveness and competitive time complexity. Experiments have been conducted to compare the CPU utilization, waiting time, round trip time and offload handling performance. Each algorithm has been evaluated based on their key performance indicators and the algorithms that performed better based on the experimental constraints were discussed.

The paper is structured as follows. An extensive review of relevant and related literature is presented in Section II. Comparative analysis has been done in section III, which includes the system model, communication and computation model. Section IV details the experimental setup used for comparison. Section V shows the outcome of the experiments. Section VI concludes the paper. Finally, future works have been presented in Section VII.

II. Literature review

A. Resource Provisioning in MEC

Resource provisioning in MEC is a challenging problem for Internet Service Providers due to the impact it has on the efficiency of the system and the Quality of service (QoS). Researchers have previously addressed this resource provisioning problem in cloud computing. However, resource provisioning in MEC is more challenging mainly because the edge servers have more resource constraints than the cloud servers and the edge servers would be deployed as a distributed environment compared to the centralized cloud. Enabling distributed computing and storage capabilities at the edge of the network will benefit delay-sensitive and computation-intensive mobile applications.

There has been a considerable amount of work done in the area of resource provisioning in MEC. Badri et al [18] have proposed a risk-based optimization for resource provisioning in MEC. In their work, they have assumed that the resource requirements of mobile applications are stochastic. Therefore, they formulated a chance-constrained stochastic program problem. They have resolved this using the Sample Average Approximation method.

Kherraf et al [9] have studied resource provisioning and workload assignment in MEC and formulated the problem as a mixed-integer program to jointly decide on the number of nodes, the location of MECs and applications to deploy. They have solved this by decomposing it into two problems, a delay aware load assignment sub-problem and dimensioning edge servers sub-problem. They have proposed optimized provisioning of edge computing resources with a heterogeneous workload in IoT networks. They concluded that the proposed tool could be used by network operators to develop cost-effective strategies for edge network planning and design.

Chang et al [19] have studied resource provisioning in MEC in the area of minimizing energy consumption of cellular networks. In their research, they investigated both the communication and computation aspect of resource provisioning to improve energy efficiency. They modelled the system as tandem queues and studied the trade-off between the subsystems on energy consumption and service latency. Based on this, they proposed an algorithm to determine the optimal provisioning of both communication and computation resources to minimize the overall energy consumption without sacrificing the performance of service latency.

Yu et al [20] have proposed a collaborative computation offloading framework for MEC. The authors have considered an offloading scenario where multiple mobile users offload duplicated computation tasks to the edge servers. Hence, creating an opportunity for edge servers to share computational results. The aim is to develop an optimal collaborative offloading strategy with data caching enhancements to reduce end-user latency. The problem has been formulated as a multi-label classification in which a Deep Supervised Learning approach has been employed to address the issue. Numerical results have shown that the proposed scheme achieves reduced delay and energy consumption compared to other schemes.

Zhou et al [21] have proposed a resource provisioning scheme for heterogeneous IoT applications on cloud-edge platforms. The scheme has been aimed at minimizing long-term operational costs while guaranteeing both hard and soft deadlines for heterogeneous IoT applications. The proposed framework employs a Lyapunov optimization technique to make online resource provisioning greedy decisions without prior knowledge of the resource statistics of the edge system. The authors have evaluated the efficiency of the proposed approach using realistic traffic and cost traces.

Ma et al [22] have proposed a mobility-aware and delay-sensitive service provisioning scheme for mobile edge cloud networks. The authors have formulated two novel optimization problems of user service request admissions with the focus of maximizing the accumulative network utility and throughput. The authors have utilized a constant approximation algorithm and an online algorithm to address the formulated problems. The authors have demonstrated the efficiency of the proposed scheme using experimental simulations.

There have been other proposals for resource provisioning techniques to offload mobile application workloads on MEC [23]. Nevertheless, none of the previous works on MEC con-

siders deadlock during offloading and resource provisioning which is a concern for distributed systems such as MEC [13].

B. Deadlock Handling Strategies

In a trusted computing scenario using edge nodes and IoT devices, high availability and reliability are crucial factors for a good user experience. Therefore, deadlock-free operations are important in achieving this goal. The absence of deadlock strategies to detect, recover or eradicate deadlock in such a system might cause deterioration of the system's performance and ineffective use of energy as deadlock might occur but the system has no way of recognizing what has happened. The standard toolset for deadlock detection is the Wait for Graph (WFG). The WFG models the relationship between the processes and the resources involved. Here, each node represents a process and an arc is originated from a process waiting for a resource to a process holding the resource. There are 4 main ways of handling deadlock. This includes (i) ignore, (ii) detect and recover, (iii) prevention and (iv) avoidance. Furthermore, there are 4 main conditions necessary for a deadlock to occur. These include (i) mutual execution, (ii) hold and wait (iii) no preemption and (iv) circular wait [12]. A simultaneous occurrence of these four conditions leads the system to an unsafe state where the system suffers from a probability of getting stuck due to unmanaged distribution of resources. Therefore, each of the deadlock handling strategies offers solutions to eradicate deadlock by ensuring that at least one of these conditions does not hold.

1) Deadlock Detection

In the design and development of a multi-threaded system, deadlock detection could be chosen as a way of handling system deadlock. If the employed algorithm detects a deadlock, the next step would be to recover the system from the deadlock. Therefore, deadlock detection and recovery go hand in hand. In this scenario, an algorithm is employed to examine the state of the system to determine if a deadlock has occurred. After this, another algorithm is used to recover the system from deadlock. To detect the presence of deadlock, a resource allocation graph and a corresponding WFG are used for a single instance for each resource type. Note that in this strategy, deadlock can happen, after which the system then detects the occurred event and attempts to recover itself. This causes an overhead of the run-time costs of maintaining the necessary information and executing the detection algorithm. Additionally, there might be potential losses inherent in recovering from a deadlock [12].

To detect deadlock for a single instance of each resource type using the WFG, an edge from E_i to E_j implies that process E_i is waiting for the process E_j to release a resource that E_i needs. The edge $E_i \rightarrow E_j$ only exists in a WFG if the corresponding resource allocation graph contains two edges $E_i \rightarrow R_q$ and $R_q \rightarrow P_j$ for some resource R_q. A deadlock exists in the system if there is a cycle in the WFG. Using this, the detection algorithm requires a runtime order of n^2 operations where n is the number of vertices in the graph. For several instances of a resource type, the runtime order to detect a deadlock would be $m \times n^2$ where m is a vector that indicates the number of available resources of each type [12].

There is extensive research on deadlock detection schemes. Farajzadeh et al [24], have proposed a distributed deadlock detection algorithm based on history-based edge chasing which resolves the deadlock as soon as its detected. According to their research, this action reduces the average persistence time of the deadlock compared to other detection algorithms. Akikazu et al [25] have proposed a deadlock detection algorithm for distributed processes. In their research, they have formulated a deadlock detection scheduling problem with the presence of system failures and derived a deadlock detection time that minimizes long-run average cost per unit time. They have concluded that the number of distributed processes and the system failure probability give a great effect on the long-run average message-complexity per unit time, but not the deadlock scheduling time. Other research on deadlock detection includes Lamport's algorithm [26] which is a mutual exclusion algorithm that uses logical clocks for event synchronization and the Chandy-Misra-Hass algorithm [27] which uses messages called probes to detect the presence of deadlock in a system.

2) Deadlock Prevention

Deadlock prevention algorithms handle deadlock in a system by trying to prevent one of the previously mentioned four conditions required for a deadlock to occur. In a typical distributed system, there is at least one non-sharable resource. Therefore, the mutual exclusion condition must hold. Due to this, deadlock cannot be prevented by denying the mutual exclusion principle. To prevent deadlock by eliminating hold and wait, two possible protocols could be used. A protocol that requires that all resources a process needs are allocated to the process before the start of execution. This will eradicate hold and wait but might lead to the under-utilization of the system. Another protocol could allow a process to request new resources only after releasing the current set of resources. However, this protocol may lead to starvation.

Deadlock can also be prevented by preempting resources from a process if the resources are required by a higher priority process. This strategy of process termination during execution is inappropriate for real-time systems in which the elapsed execution time of the process must be predictable [28]. The final method of preventing deadlock is by eliminating the circular wait condition. To ensure that this condition never holds, a protocol can be used to impose a total ordering of all resource types and require that each process requests resources in increasing order of enumeration. Example, if $R = \{R_1, R_2..R_z\}$ is a set of resource types which has been assigned unique integer numbers from 1 to z. A process can only request resources in increasing order of enumeration. Therefore, if a process request R_i then it can only request for another resource $R_j \Leftrightarrow F(R_j) > F(R_i)$ [12] here $F(x)$ is the order of enumeration of x.

There have been many deadlock prevention strategies proposed by different researchers in different computer science fields to eradicate deadlock by preventing one of the necessary four conditions required for a deadlock to occur. In the field of Service-Oriented Architecture (SOA) infrastructure, Lin Lou et al [29] have proposed a deadlock prevention strategy to eradicate the possibility of deadlock caused by resource

locking based on two-phase commit protocol. This requires that each transaction obtains all needed locks before the second commit phase. To solve this, they have utilized a timestamp-based restart policy for global resource allocation.

In the context of web SOA where the competition of web resources by web services could lead to deadlock. Ding et al [30] have proposed a method to analyze and verify the deadlock prevention solutions using trace semantics of communicating sequential processes. The proposed formal modelling approach has proved useful in the verification of deadlock solutions analyzed in the paper. Furthermore, in the context of Grid systems with resource sharing capabilities, simultaneous requests of co-allocation of resources by multiple applications could lead to deadlock. To address this problem, Chuanfu et al have [31] proposed a deadlock prevention method for fast allocation of grid resources based on an atomic transaction. Utilizing this method, all resources required by a process at the time of the request are specified. The request succeeds if all the resources required are available.

In this research, two preventive algorithms that have been investigated are wound-wait and wait-die. Both algorithms use timestamp-based techniques and they favour the older processes with an older timestamp. These algorithms have been used due to their efficiency, competitive time complexity and practical applications, especially in database systems [32].

a) Wound-wait algorithm: Wound-wait deadlock prevention algorithm is a non-preemptive technique. Here, when an older process requests a resource that is currently held by a younger process, the younger process is rolled back. However, when a younger process requests a resource that is held by an older process, the younger process waits. If P_i and P_j are both processes and P_i requests for a resource held by P_j. Then P_i is rolled back if $t(P_i) > t(P_j)$ i.e P_i is younger, else P_i can wait. Here, $t(P_i)$ and $t(P_j)$ are timestamps.

b) Wait-Die algorithm: The wait-die deadlock prevention algorithm is a preemptive technique. In this scenario, when an older process requests a resource that is held by a younger process, the older process waits. However, when a younger process requests a process that is held by an older process, it dies. If P_i and P_j are both processes and P_i requests for a resource held by P_j. Then P_j is rolled back if $t(P_i) < t(P_j)$ i.e P_i is older. Where $t(P_i)$ and $t(P_j)$ are timestamps, else P_i can wait.

3) Deadlock Avoidance

The drawbacks of using a deadlock prevention method include low device utilization and reduced system throughput. Alternatively, the deadlock avoidance mechanism could be used. In contrast to the prevention method, this works by requiring additional information on the complete sequence of the resource request. With this prior information on the requests and resources, the system decides if a process should run or wait with the motive of avoiding a possible deadlock. For the avoidance method to work, the simplest model requires that the maximum amount of resources for each resource type is declared. With these data, the model ensures that a circular-wait condition never exists during the dynamic resource allocation. Any potentially unsafe resource request is denied. The system is said to be in a *safe state* if the maximum number of resources requested for each process can be allocated and there exists no possible sequence of future requests in deadlock. If a safe sequence exists, then the system is said to be in a safe state. There is a safe sequence of processes $[P_1, P_2, P_3...P_n]$, if the resource request that would be made by each process P_i can be satisfied by the currently available resources including resources held by all P_j with $j < i$ [12]. A system is said to be in an *unsafe state* if it is not guaranteed that all possible sequences of future requests will not lead to a deadlock. Not all unsafe states are deadlocks, but an unsafe state may lead to a potential deadlock in a system. There are two well-known deadlock avoidance algorithms which are the resource allocation graph algorithm and the banker's algorithm.

a) Resource Allocation Graph: The resource allocation graph is only used if the resource allocation system has only one instance of each resource type. While using the resource allocation graph for deadlock avoidance, if a process P_i requests for a resource R_j,$(P_i \rightarrow R_j)$, the request is only granted if $R_j \rightarrow P_i$ does not lead to a cycle in the resource-allocation graph. Safeness of the system is checked by using a cycle-detection algorithm which requires an order of n^2 operations where n is the number of processes in the system.

b) Banker's Algorithm: Resource-allocation graph cannot be applied to a resource allocation system with multiple instances of each resource type due to limitations. However, Banker's algorithm could be used. If the Banker's algorithm is applied, then each process must declare the maximum amount of resources for each resource type that it will require to complete execution. This declared number must not exceed the total amount for each resource type in the system, else the system would be in an unsafe state. Four data structures must be maintained while using the Banker's algorithm.

- **Available**: This is a vector of the number of available resources for each resource type. The length of the vector is m where m is the number of available resources.
- **Max**: This is a matrix of the maximum resource demand for each process. The matrix size is mn where m is the number of available resources and n is the number of processes.
- **Allocation**: This is a matrix of the number of resources currently allocated for each process. The matrix size is mn where m is the number of available resources and n is the number of processes.
- **Need**: This is a matrix of the remaining amount of resources that each process needs. The matrix size is mn where m is the number of available resources and n is the number of processes. $Need = Max + Allocation$

The time complexity to determine if a state is safe or not is mn^2 [12]. There has been a great deal of research done on the improvement of banker's algorithm over the years. In each case, the algorithm is extended, improved or applied in a different area in computer science. The most notable adjustments have been made in 1999 [33], 2000 [34] and 2006 [35]. Sheau-Dong Lang [33] has assumed that the control flow of the resource-related calls of processes forms rooted trees. Based on this, a quadratic-time algorithm has been proposed. The algorithm decomposes trees into regions and computes

the associated maximum resource claims before the process execution. The information collected is used at runtime to verify the safety of the system using the original banker's algorithm. Tricas et al [34] have applied Banker's algorithm in the field of flexible manufacturing systems. They have modelled the problem employing Petri nets and proposed two improvements based on the knowledge of process structure. Their research has proven that the improved algorithm has much more concurrency than the original banker's algorithm. Other deadlock avoidance algorithms that have been developed includes the graphical deadlock avoidance algorithm proposed by El-Kafrawy [36]. The improvement solves the deadlock avoidance problem in sequential resource allocation systems using a polynomial graphical solution. The graph updates dynamically each time a new resource is requested. Another example is the deadlock avoidance algorithm for streaming applications proposed by Li et al [37] using both a propagating algorithm and a non-propagating algorithm.

C. Real-Time Scheduling

In a trusted computing environment where high availability and reliability are important factors, often there is a specific response deadline time constraint that the system must meet. If this is the case, the system is said to be a real-time system. The system may or may not meet this time demand. This depends mainly on the capacity of the system to perform computations at a given time. In a real-time application, there are multiple tasks with different criticality levels. The tasks could either be soft real-time, hard-real-time or firm real-time. For a given set of tasks $T = \{t_1, t_2, t_3 ... t_n\}$. Task t_i is said to be a hard real-time task if the execution of t_i must be completed by a given deadline D_i and $W_i \leq D_i$ where W_i is the worst-case execution time of t_i. Task t_i is said to be a soft real-time task if the penalty it pays increases as the r_i increases. Here, r_i is the time elapsed between the deadline of t_i and the actual completion time. The penalty function $P(t_i) = 0$ if $W_i \leq D_i$ else $P(t_i) > 0$. The task t_i is said to be a firm real-time task if an increase in reward depends on how early t_i finishes its computation before the given deadline D_i. The Reward function $R(t_i) = 0$ if $W_i \geq D_i$ else $R(t_i) > 0$. In this research, two optimal Real-time scheduling algorithms have been studied. These are Rate Monotonic Scheduling Algorithm (RMS) and the Earliest deadline First algorithm (EDF). These algorithms were selected because they are well-known baseline scheduling algorithms for real-time systems [38] and they also have a competitive time complexity.

1) Rate Monotonic Scheduling Algorithm

The Rate Monotonic Scheduling (RMS) Algorithm is a priority-driven algorithm with priorities well known before the arrival of the task. These priorities are determined by the time period of each task and are the same for all instances of the same task. RMS is the most widely used and studied real-time algorithm [38]. Some assumptions are made while using the RMS algorithm, these assumptions include: (i) The tasks have no precedence constraints and all tasks are independent. (ii) it is assumed that only processing requirements are significant. (iii) it is assumed that the tasks have no non-preemptable section and the cost of preemption is negligible. (iv) it is also assumed that the tasks are periodic and that t_i has a higher priority than $t_j \Leftrightarrow i > j$. The shorter the period, the higher the priority. If a lower priority task t_j is running and a higher priority task t_i is waiting to run, t_i will preempt t_j. RMS assigns higher priority to tasks that use the CPU more often. The algorithm complexity is $n(2^{1/n} - 1)$. RMS is referred to as an optimal real-time algorithm because if a given set of processes cannot be scheduled by RMS, then it cannot be scheduled by any other algorithm that uses static priorities [12].

2) Earliest Deadline First Algorithm

The Earliest Deadline First (EDF) algorithm [38] is also a priority-driven algorithm which assigns priorities according to tasks deadline. EDF gives a higher priority to a task t_i that has an earlier deadline d_i. t_i will always preempt a task with a lower priority t_j which have a higher deadline d_j. EDF uses a dynamic priority assignment. The priority of the tasks are assigned as the tasks arrive based on the task's deadline requirements. The priorities of other tasks are adjusted to reflect the deadline of newly runnable processes. The following assumptions are made when using the EDF scheduling algorithm: (i) The tasks have no precedence constraints and all tasks are independent. (ii) it is assumed that only processing requirements are significant. (iii) it is assumed that the tasks have no non-preemptable section and the cost of preemption is negligible. The EDF algorithm has a worst-case runtime of $O((N + \alpha)^2)$ where α is the number of aperiodic tasks and N is the total number of requests in each hyper-period of n periodic tasks in the system [39]. EDF is referred to as an optimal uniprocessor real-time scheduling algorithm because it schedules tasks so that they meet their deadline requirement with 100% CPU utilization. If EDF cannot feasibly schedule a set of tasks on a uniprocessor then no other algorithm can. This is proved using the time slice swapping technique [38].

D. Research Contributions

The main contributions of this paper are listed as follows:

- Using a case study for AVs as a real-time system, to compare how deadlock avoidance and prevention mechanisms will perform in real-time scenarios using RMS or EDF in prioritizing workloads. In this analysis, different metrics have been considered including Round-trip time, Queue waiting time, CPU utilization and Ratio of Local execution to collaborative MEC to cloud.
- An emulation testbed for multi-access mobile access computing and Software-defined Networking based on the GNS3 platform for experimental purposes.
- A deadlock aware Collaborative decision offloading scheme for MECs during resource provisioning in AVs.

III. COMPARATIVE ANALYSIS

In this paper, a comparative study is carried out for deadlock avoidance and deadlock prevention algorithms for multi-access mobile edge computing environments. Here 6 case study algorithms have been considered based on the structure proposed in our previous study on deadlock in multi-access mobile edge

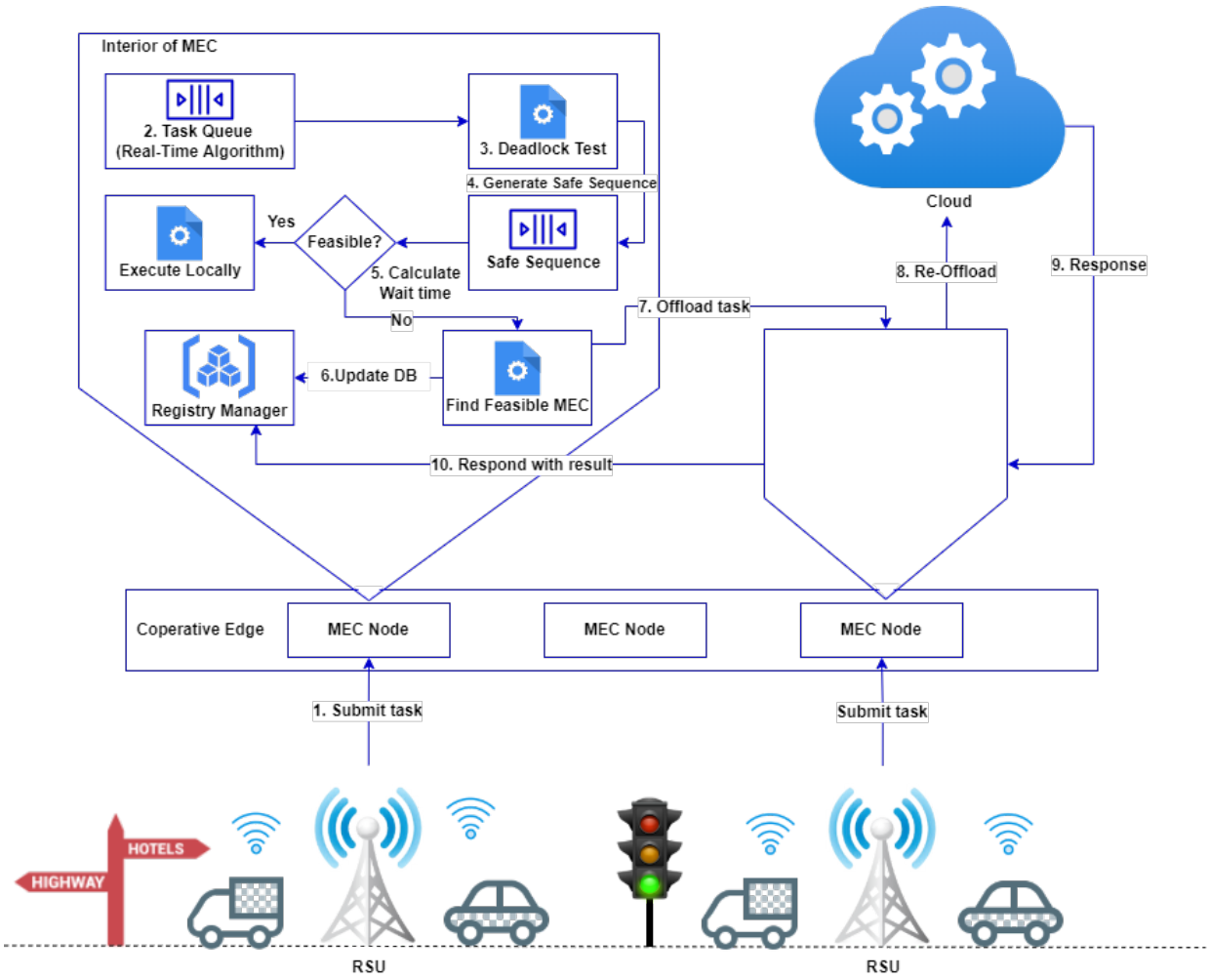

Fig. 1. The adapted algorithm workflow structure for modelling the task sequence from the end device to the MEC node for processing.

TABLE I
ALGORITHMS USED DURING EXPERIMENT

Deadlock Prevention Schemes	Wait-die Algorithm
	Wound Wait algorithm
Deadlock Avoidance scheme	Bankers resource request algorithm
Realtime scheduling Schemes	Rate monotonic scheduling algorithm
	Earliest Deadline First Algorithm

TABLE II
COMPARED ALGORITHMS

Alias	Compared Algorithms
ALG_1	RMS and Banker algorithm
ALG_2	EDF and Banker algorithm
ALG_3	RMS and Wound Wait
ALG_4	RMS and Wait die
ALG_5	EDF and Wound Wait
ALG_6	EDF and Wait Die

computing for industrial IoT [13]. Each compared algorithm is composed of a deadlock algorithm and a real-time scheduling algorithm. The algorithms used for this design can be seen in Table I.

The 6 compared algorithms that have been produced as a result can be seen in Table II. The algorithm workflow for each of the six algorithms is the same structure as is in Figure 1. Tasks are sent from the RSU to the local edge node for resource provisioning. In the MEC node, tasks are put into a job queue and the queue is prioritized using a real-time scheduling algorithm. Then a deadlock algorithm is employed to reduce or eradicate the chances of deadlock. Thereafter, the waiting time is calculated for each task received and an assumed finishing time P_{t_i} for each task t_i is predicted. If $P_{t_i} < D_{t_i}$ where D_{t_i} is the deadline for task t_i, then a MEC M_i is identified that meets the deadline requirement using a collaborative decision algorithm. This algorithm is detailed in subsection F. If such MEC is not found, then the tasks are sent to the central cloud to be executed and the MEC node acts as a proxy. For each of the compared algorithm, this structure remains the same but an appropriate real-time algorithm and deadlock algorithm is utilized. Further details about the algorithm structure are detailed in our previous study [13].

A. Deadlock in distributed MEC

To describe the deadlock condition in distributed MEC, let's assume a set of processes $P = \{p_1, p_2.., p_n\}$ and a set of resources $R = \{r_1, r_2.., r_m\}$, where n and m are

TABLE III
MEANING OF PARAMETERS

Notation	Meaning
M	Denotes a cluster of MEC nodes
M_j	Denotes a MEC node
U	Set of end devices
u_i	Denotes end device
W_{u_i}	The total workload for an end device u_i
$T_j^{u_i}$	The task for the end device u_i
REQ	Requirement vector
S_{M_j}	CPU frequency
f_i	The execution time of the task T_i^u
WT	Waiting time
K_i	Processing Delay
H_i	Communication cost
$trans_i$	Transmission delay
$prop_i$	Propagation delay
tr_{M_j}	Transmission rate of M_j
RES	Required resource type
$\|RT\|$	number of resource-type
MAX	max resource for each RES
t_i	time period for EDF
D_i	Deadline
E	Constraints for EDF
W_w	Constraints for wound-wait
W_d	Constraints for wait-die
$BA(x)$	Bankers algorithm function
TC	Total Time cost $(K_i + H_i)$
M_j^s	Edge status vector
$Cl_{M_j}^{status}$	set of all M_j^s in the edge cluster

the number of processes and resources respectively. These resources and processes are present in the collaborative MEC space. However, they might not reside in the same MEC. Deadlock occurs if a process p_i is waiting for a resource r_a that is currently held by another process p_j. Additionally, p_j is waiting for a resource r_b that is currently held by p_i. If neither p_i nor p_j can be preempted while in waiting state, the system would be in a deadlock.

B. System Model

In this research, a distributed architecture which consists of a pool of MEC nodes is considered as a platform for resource provisioning. Let's consider a cluster of edge servers. A finite non-empty set of edge servers in the same cluster is denoted as $M = \{M_1, M_2.., M_n\}$. Let's assume that a finite non-empty set of end devices $U = \{u_1, u_2.., u_n\}$ are connected to the edge network such that $u_i \in U$ and $M_j \in M$ maintains a disjoint many-to-one cardinality. Here an edge server is connected to many end devices, but no end device is connected to multiple edge server. Each u_i has a workload $W_{u_i} = [T_1^{u_i}, T_2^{u_i}, ..T_n^{u_i}]$ which contains an array of tasks to be executed. For each $T_j^{u_i}$ in W_{u_i} the u_i computes an offloading decision $a_j \in \{0, 1\}$, where $a_j = 0, a_j = 1$ represents "execute locally" and "offload", respectively. It is assumed that the end device makes an offloading decision based on its battery life and computational resources. Let's assume that each u_i is connected to the closest M_j and hence offloads all $T_j^{u_i} \in W_{u_i}$ s.t $a_j = 1$. For each $T_j^{u_i}$ that is offloaded, the u_i also sends a requirement vector $REQ = \{c_i m_i, l_i, s_i\}$. REQ is characterized by number of CPU cycles, memory, maximum latency and data size respectively.

C. Computational model

Let's denote the computation capacity of each MEC M_j in M as S_{M_j}. This is the CPU frequency. Let's assume that each M_j maintains a queue $Q_{M_j} = [T_1^u, T_2^u..T_n^u]$ of tasks offloaded to M_j. The execution time of a task T_i^u offloaded to M_j is

$$f_i = \frac{c_i}{S_{M_j}} \tag{1}$$

The waiting time for a newly added task T_{n+1}^u is

$$WT_n = \sum_{i=1}^{n} f_i \tag{2}$$

Therefore, the total processing delay K_i for T_i^u is

$$K_i = WT_n + f_i \tag{3}$$

D. Communication model

For a task T_i^u offloaded to an edge node M_j, the communication cost of offloading the task H_i can be expressed as the following

$$H_i = trans_i + prop_i \tag{4}$$

Where $trans_i$ and $prop_i$ are the transmission delay and propagation delay respectively. The transmission delay can be expressed as

$$trans_i = \frac{s_i}{tr_{M_j}} \tag{5}$$

Where tr_{M_j} is the transmission rate of M_j. Substituting eq (5) in eq (4), the communication cost can be expressed as

$$H_i = \frac{s_i + (prop_i \times tr_{M_j})}{tr_{M_j}} \tag{6}$$

E. Modelling of the algorithms in table II

While using any of the algorithms in Table II, an identification id is required to uniquely identify each process. To define the requirements for a set of processes P, for any of the defined algorithms, the variable constraints for the algorithm components are first defined.

1) Bankers Algorithm

In using the Banker's algorithm, for each process sent to the MEC for resource provisioning, two vectors are required. These are the resource type required RES by the process and the maximum resource for each resource type MAX. Therefore, for each process the resource type constraint

$$RES = \{res_i \ \in 0, 1 \mid i \in \{0...(|RT| - 1)\}\} \tag{7}$$

where, $|RT|$ is the number of resource-type. When $r_i = 0$, the resource type at the i^{th} position is not required otherwise $r_i = 1$ specifies that the resource type is required. It is assumed there are three resource types (CPU, Memory and Storage) that can be claimed by each process. Likewise, the maximum resource for each resource type is defined as

$$MAX = \{max_i \mid \forall\, i \in RES_r\} \tag{8}$$

where

$$RES_r = \{res_i \in RES \mid res_i = 1\} \subseteq RES \tag{9}$$

2) RMS

While using the RMS algorithm, for each process that is sent to the MEC for resource provisioning the computation time or capacity C_i and the time period t_i will be required.

$$t_i = \left[\frac{1}{f_i} \mid \forall\, i \in \{1..|P|\}\right] \tag{10}$$

Therefore, variable constraints needed for RMS

$$REQ_i = \{[C_i, t_i] \mid \forall\, i \in \{1..|P|\}\} \tag{11}$$

3) EDF

While using the EDF algorithm, for each process sent to the MEC for resource provisioning the computation time or capacity C_i, time period t_i and the deadline D_i will be required. Therefore, variable constraints needed for EDF

$$E = \{[C_i, t_i, D_i] \mid \forall\, i \in \{1..|P|\}\} \tag{12}$$

4) Wound-wait /Wait die

For each process sent to the MEC for resource provisioning while using the wound-wait or wait die, the resource type required RES and the time stamps Ts for each process are required. Here, RES is obtained the same way as in eq 7 and $Ts = \{t_i \mid i \,\forall\, \{1...|P|\}\}$. Therefore, the variable constraints for Wound wait W_w and Wait die W_d

$$W_w = W_d = \{RES_i, Ts_i \mid i \,\forall\, \{1...|P|\}\} \tag{13}$$

F. Deadlock constraint of the algorithms in Table II

1) Rate Monotonic Scheduling and Banker algorithm

In this algorithm for a set of processes P, combining 7 and 9

$$P = \{P_i \mid \forall\, P_i \in (id, RES, MAX, REQ)\} \tag{14}$$

Let $RMS(x)$ and $BA(x)$ be functions of RMS and Banker's algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \tag{15}$$

Where, P_{rt} is a set of tasks that can be executed in real-time. Then putting P_{rt} in the Banker's function,

$$BA(P_{rt}) \rightarrow P_{safe} \tag{16}$$

Where, P_{safe} is the deadlock-free safe sequence.

2) Earliest Deadline First and Banker algorithm

In this algorithm for a set of processes P, combining 7, 9 and 12

$$P = \{P_i \mid \forall\, P_i \in (id, RES, MAX, E)\} \tag{17}$$

Let $EDF(x)$ and $BA(x)$ be a function of the EDF algorithm and Banker's algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \tag{18}$$

Where, P_rt is a set of tasks that can be executed in real-time.

$$BA(P_{rt}) \rightarrow P_{safe} \tag{19}$$

Where, P_{safe} is the deadlock-free safe sequence.

3) Rate Monotonic Scheduling and Wound Wait

In this algorithm for a set of processes P, combining equations 11 and 13

$$P = \{P_i \mid \forall\, P_i \in (id, W_w, REQ)\} \tag{20}$$

Let $RMS(x)$ and $W_W(x)$ be a function of the RMS algorithm and Wound wait algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \tag{21}$$

Where, P_{rt} is a set of tasks that can be executed in real-time.

$$W_w(P_{rt}) \rightarrow P_{safe} \tag{22}$$

Where, P_{safe} is the deadlock-free safe sequence.

4) Rate Monotonic Scheduling and Wait die

In this algorithm for a set of processes P, combining equations 11 and 13

$$P = \{P_i \mid \forall\, P_i \in (id, W_d, REQ)\} \tag{23}$$

Let $RMS(x)$ and $W_d(x)$ be a function of the RMS algorithm and Wound wait algorithm respectively. Then,

$$RMS(P) \rightarrow P_{rt} \subseteq P \tag{24}$$

Where, P_{rt} is a set of tasks that can be executed in real-time.

$$W_d(P_{rt}) \rightarrow P_{safe} \tag{25}$$

5) Earliest Deadline First and Wound Wait

In this algorithm for a set of processes P, combining equations 12 and 13

$$P = \{P_i \mid \forall\, P_i \in (id, W_w, E)\} \tag{26}$$

Let $f : EDF(x)$ and $f : W_w(x)$ be a function of the EDF algorithm and Wound wait algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \tag{27}$$

Where, P_{rt} is a set of tasks that can be executed in real-time.

$$W_w(P_{rt}) \rightarrow P_{safe} \tag{28}$$

Where, P_{safe} is the deadlock-free safe sequence.

6) Earliest deadline First and Wait Die

In this algorithm for a set of processes P, combining equations 12 and 13

$$P = \{P_i \mid \forall\ P_i \in (id, W_d, E)\} \tag{29}$$

Let $f : EDF(x)$ and $f : W_d(x)$ be a function of the EDF algorithm and Wait die algorithm respectively. Then,

$$EDF(P) \rightarrow P_{rt} \subseteq P \tag{30}$$

Where, P_{rt} is a set of tasks that can be executed in real-time.

$$W_d(P_{rt}) \rightarrow P_{safe} \tag{31}$$

Where, P_{safe} is the deadlock-free safe sequence.

G. Collaborative offloading decision

In this sub-section, the collaborative offloading decision making for the proposed algorithm has been described. In the proposed algorithm, the offloading decision is made by considering the time constraint of a task T_i^u and the deadlock constraint of the MEC resource. For the time constraint, the following must be satisfied for execution

$$TC < l_i \tag{32}$$

Where TC is the time cost and is a summation of the computational cost and communication cost.

$$K_i + H_i < l_i \tag{33}$$

Substituting 3 and 6 in 33

$$WT_n + f_i + \frac{s_i + (prop_i \times tr_{M_j})}{tr_{M_j}} < l_i \tag{34}$$

The deadlock constraint is determined using one of the algorithm models in the previous section. For simplicity let's assume that the constraint is determined by the $BA(x)$ in 16. $BA(x)$ returns a safe sequence or false if the system is not in a safe state. M_j makes an offloading decision a_i for each newly added task T_i^u. $a_i \in \{0, 1, 2\}$, where $a_i = 0$, means execute locally, $a_i = 1$ means send to another edge node and $a_i = 2$ means offload to the central cloud.

$$(TC < l_i)\ and\ (BA(x) \rightarrow Safe) \tag{35}$$

If 35 is True, then $a_i = 0$. Otherwise, an MEC that fits the description is sort after. If such MEC exists, then T_i^u is offloaded to it else it is offloaded to the cloud. To ensure that an edge node M_j can calculate 35 for another edge node M_k, each M_j multicasts its status M_j^s to the MEC cluster after each update to WT_n. To reduce the communication overhead, the number of MECs in a cluster is minimized.

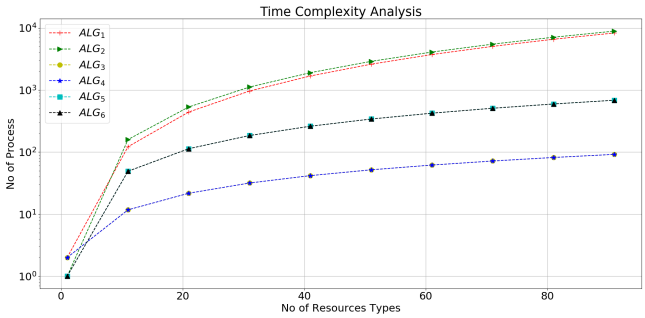

Fig. 2. Time Complexity Analysis for algorithms in Table II

TABLE IV
TIME COMPLEXITY COMPARISON OF EACH COMPONENT ALGORITHM

Algorithms		Complexity
Deadlock	Banker's algorithm	$O(mn^2)$
	Wound wait	$O(mn)$
	Wait-die	$O(mn)$
Scheduling	RMS	$O\left(n(2^{1/n} - 1)\right)$
	EDF	$O(n \log_2 n)$

$$M_j^s = \{S_{M_j}, WT_n, Mem_{M_j}, \} \tag{36}$$

Where Mem_{M_j} is the memory utilization of M_j. Therefore, each M_j maintains the following

$$Cl_{M_j}^{status} = \{M_1^s, M_2^s, ..M_n^s\} \tag{37}$$

The collaborative algorithm is presented in algorithm 1

H. Time Complexity

The differences in the time complexity of the algorithms used in this research to design each algorithm can be seen in Table II. Comparing the deadlock algorithms, banker's algorithm has the highest order of time complexity. However, comparing the scheduling algorithms, the EDF algorithm has the highest order of complexity. The Time complexity graph in Figure 2 compares the time complexity of each of the compared algorithms. The graphs show the scalability of each of the compared algorithms with an increase in the number of processes and resources. The graph illustrates that ALG_1 and ALG_2 are the most scalable case study algorithms with increase in the number of processes and number of resource types. On the other hand, ALG_3 and ALG_4 is the least scalable algorithm out of the six algorithms.

IV. EXPERIMENTAL SETUP

In this section, the experimental setup is presented and how the compared algorithms are tested is outlined. The components that make up the system and the associated tools are discussed. The objective of this section is to evaluate the performance of the algorithms using two different environmental setups and evaluate the results obtained to better understand the strengths and limitations of the algorithms.

This section is broken down into two subsections as two different experimental setups have been used to evaluate and

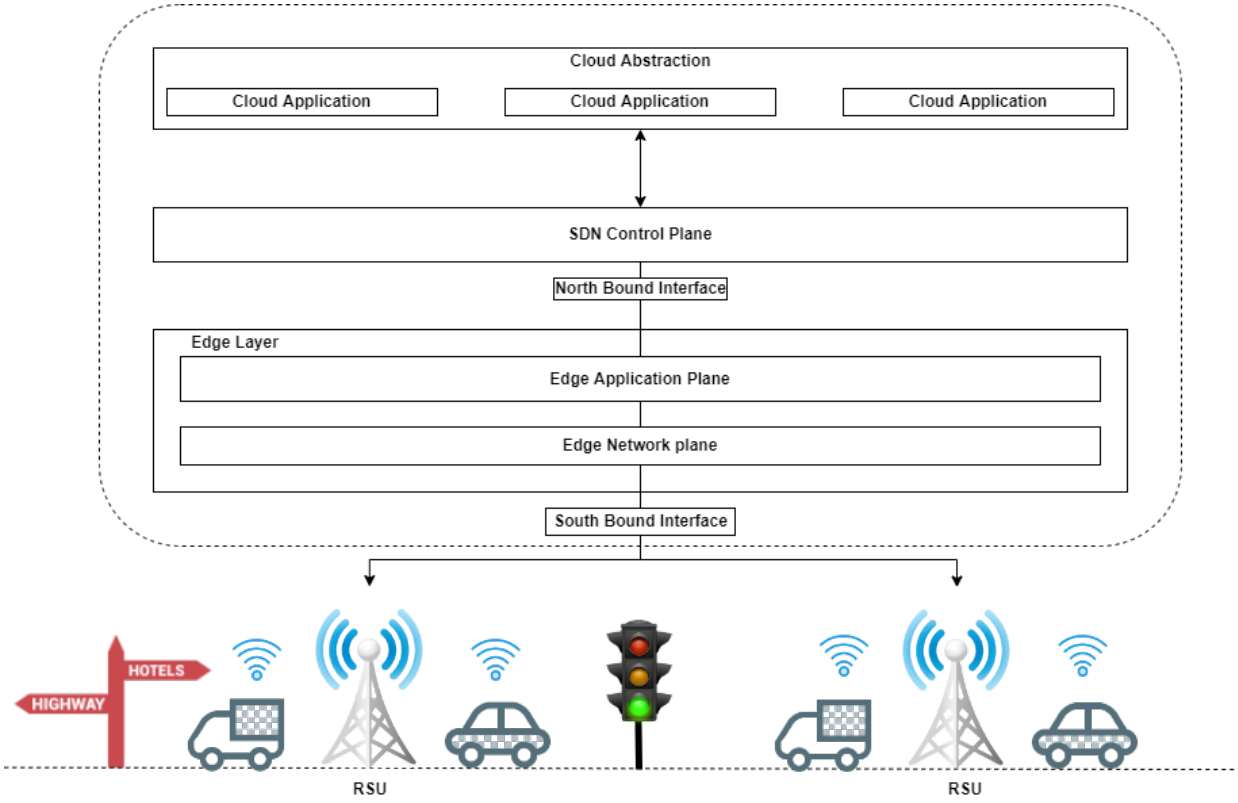

Fig. 3. High Level Deployment Architecture that has been adapted for experimentation. The figure consists of the RSU, edge components, the SDN control plane and the cloud abstraction.

Algorithm 1 Collaborative Offloading Decision Algorithm

Input $T_i^u, \{c_i m_i, l_i, s_i\}, M_j^s, \{S_{M_j}, WT_n, Mem_{M_j},\}$
Output a_i

$tk \leftarrow ((TC < l_i)\ and\ (BA(x) \rightarrow Safe))$
if $tk = True$ **then**
  $a_i \leftarrow 0$
  return a_i
else
  while M_j in $Cl_{M_j}^{status}$ **do**
    Substitute M_j^s and determine tk
    if tk = True **then**
      $a_i \leftarrow 1$
      return a_i
    end if
  end while
  if suitable M_j not found **then**
    $a_i \leftarrow 2$
    return a_i
  end if
end if

compare the case study algorithms. Both experiments have been carried out using Graphical Network Simulator-3 (GNS3) platform [40]. GNS3 is a network software emulator first released in 2008 that can be used to emulate complex networks with a combination of virtual and real devices. GNS3 has been used because it provides a platform to emulate real networks using virtualization concepts in contrast to simulation platforms like *cloudsim* [41] or *ns-3* [42].

A. Deployment Architecture

Figure 3 illustrates the high-level diagram of the experimental deployment. The Edge layer consists of the network and application plane. The Edge layer extends the conventional infrastructure by providing compute and storage capacities to the RSU for resource provisioning. Compute and storage decisions made in the Edge layer are made through the edge application plane. The Edge servers in the MEC layer collaborate among themselves using the network plane to support the demand from the IoT devices/UE. Routing and forwarding decisions are made by the SDN controller in the control plane. The MEC uses multicast for cooperative communication. In the experimental setup, MQTT broker-based [39] multi-cast communication is used. MQTT is used to simulate multicast cooperative communication between the RSUs during experiments. Other cooperative communications mediums could also be used. Comparisons between these mediums are outside the scope of this research. The RSU reach the edge layer through a cellular communication link. Tasks are sent to the cloud if they cannot be scheduled in the edge layer. In the experimental setup in GNS3 platform [43], the MEC leverages the cloud-native philosophy using containerised Open VSwitch hosts communicating through an SDN controller. Each of the algorithms is implemented using *python* [43] and is deployed as an MEC service on each MEC

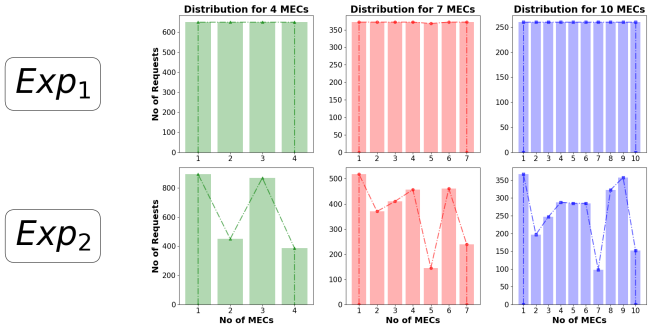

Fig. 4. Task Distribution adapted for Exp1 and Exp 2

node. To avoid time stealing during the experimentation, the CPU and Memory utilization of each MEC host has been configured to be limited to the Table V. The experiment has been conducted using two physical compute nodes whose specifications can also be seen in the Table V. Each of the compute nodes has an Intel(R) Core(TM) i7-8550U processor. The compute 1 (*C1*) runs the GNS3 emulator software and GNS3 VM1 while GNS3 VM2 runs on compute 2 (*C2*).

TABLE V
SYSTEM SPECIFICATIONS DURING EXPERIMENTS

Name	Operating System	CPU cores	Memory
Compute1 (C1)	Windows (x64)	16	32GB
Compute2 (C2)	Windows (x64)	8	16GB
MEC	Ubuntu (x64)	0.5	512MB
End device	Alpine (x64)	0.4	400MB
GNS3 VM1 (C1)	Ubuntu 18.04 (x64)	8	16GB
GNS3 VM2 (C2)	Ubuntu 18.04 (x64)	4	8GB
Software	**GNS3 Emulator**		

During the experiment, each node in the MEC layer runs one of the algorithms listed in Table II. The end devices are emulated using ubuntu linux containers. The end device generates tasks based on predefined experimental task profiles and sent to the MEC for processing. The task profile includes the CPU, memory, data size and latency constraints. The MEC then applies the scheduling algorithm followed by the deadlock algorithm for each task. The task is either executed locally, re-offloaded to another MEC or cloud depending on the result from Algorithm 1. A cluster of linux servers running the task processing application simulates the cloud service.

A pair of experimental setups have been used to evaluate the algorithms. For each of the setups, the experiment has been conducted with 4, 7 and 10 MECs. Additionally, each MEC receive 2600 requests where each request contains $|T_i|$ number of tasks where, $|T_i| \in \{1, 2, ..n\}$. The higher the value of n, the more load on the MEC and the more difficult it is to meet the deadline. n has been set to 3 in the following experiments. The task arrival at each MEC node is assumed to follow the Poisson arrival process with a varying arrival rate $\lambda_t (t \in \{1, 2..n\})$. The difference between the two experimental setups is the client request distribution.

B. Experimental Setup 1 (Exp1)

In this experiment, each MEC node receives the same number of total requests in each run. Figure 4 Exp_1 depicts the request distribution used for experiment setup 1. This setup simulates a scenario during co-operative offload where MECs are equally busy.

C. Experimental Setup 2 (Exp2)

In this experiment, each MEC node receives an unequal amount of requests in each run. Figure 4 Exp_2 depicts the request distribution used for experiment setup 2. This setup simulates a scenario during co-operative offload where MECs are unequally busy.

V. PERFORMANCE COMPARISON RESULTS

In this section, the performance of the compared algorithms obtained during the experiment has been evaluated. The metrics used for these comparisons are listed below.

- CPU Utilization of the MEC node
- Round Trip time
- The waiting Time
- The ratio of tasks re-offloaded or executed locally on the MEC

A. Experimental Setup 1 Results

This section contains experimental results obtained during experimentation using the Exp_1 setup.

1) CPU Comparison

The CPU utilization for each of the algorithm obtained after the experiments have been presented in Figure 5. From the results obtained it can be deduced that the CPU utilization of the MEC platform decreases gradually with an increase in MEC nodes. This occurs because the task loads are balanced among MECs while the number of tasks remains the same. Thereby, reducing the number of tasks processed per MEC node. The ALG_4 achieves the best CPU utilization convergence as depicted in Figure 5. The least convergence CPU utilization is the ALG_1.

2) Round Trip Time

Figure 6 shows the round-trip time comparison of each algorithm during the experiment. Each MEC (M_i), periodically monitors the round-trip time to reach each of its neighbouring MECs. M_i would not check the round-trip time (RTT) to itself, therefore each MEC would monitor $N-1$ MEC nodes during the experimentation. Where N is the total number of MECs in the cluster. The RTT is used to approximate the communication delay between two MECs. The round-trip time ranged from an average of 0.89 to 2.13 milliseconds.

3) Waiting Time

The waiting time is crucial during the algorithm run time because it is one of the factors used by each MEC to determine which of the neighbouring MEC is suitable for task re-offload if need be. The waiting time here is obtained for each of the MEC periodically, similar to how the RTT is obtained. The waiting time $WT_{m_i \to m_j} = rtt_{m_i \to m_j} + Q_{m_j}$. Where $rtt_{m_i \to m_j}$ is the round-trip time from M_i to M_j and Q_{m_j} is the queue waiting time for M_j. During reoffloading, a MEC with the lowest waiting time is always selected to evenly balance the load across the MEC platform. The waiting time

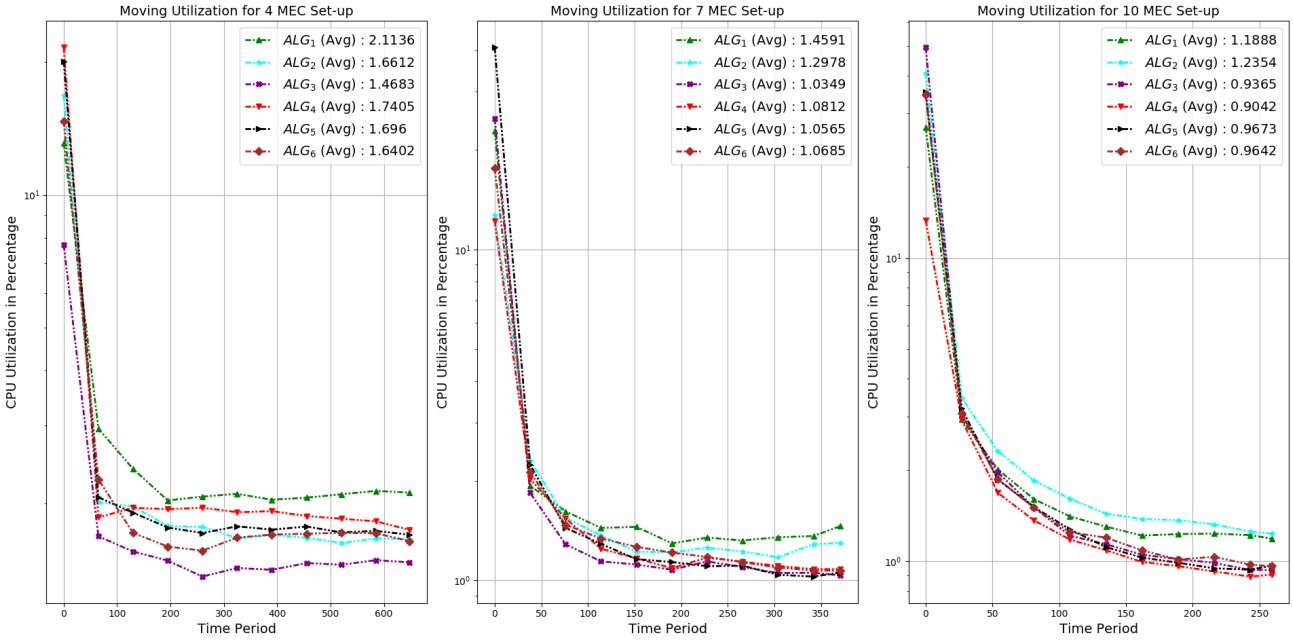

Fig. 5. The CPU utilization for each of the algorithm during the experiments in percentage and the average CPU utilization for Exp1 is displayed.

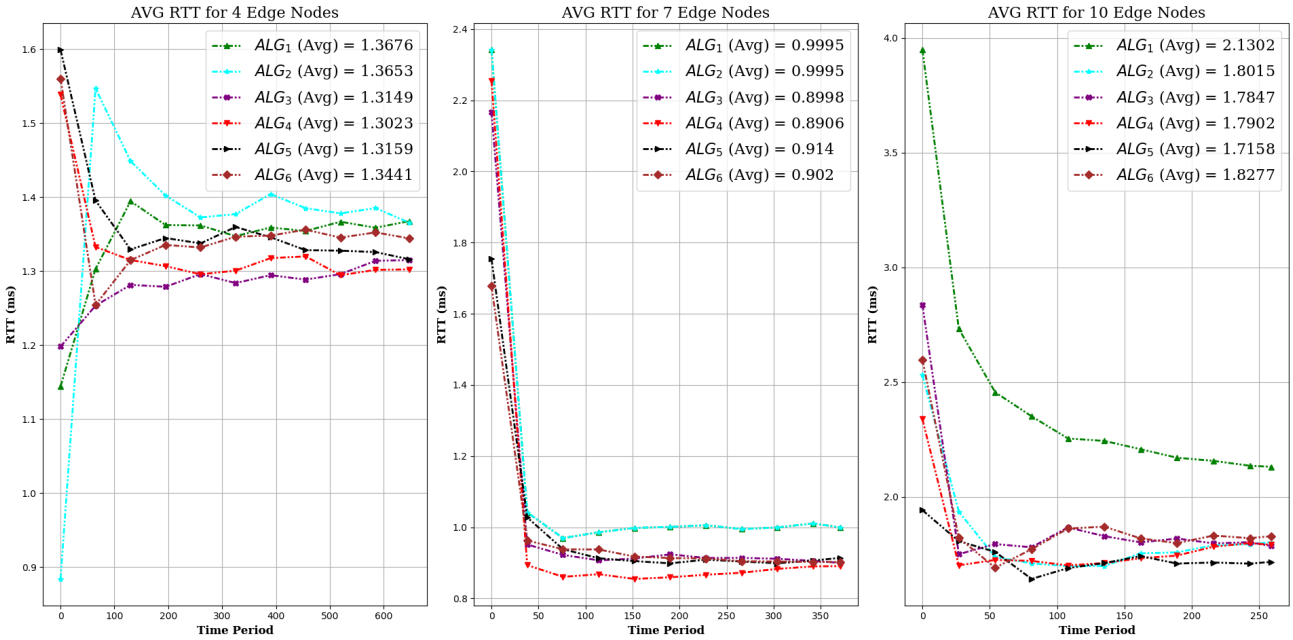

Fig. 6. The RTT obtained for each of the algorithm during the experiments in milliseconds and the average RTT for Exp1 is displayed.

for all six algorithms shown in Figure 7 for each of the experimental runs is approximately the same which guarantees experiment fairness. Averaging the results of the 3 sub experiments with 4, 7 and 10 MECs, ALG_4 achieves the lowest waiting time.

4) Offload vs Local

The comparison between the ratio of tasks re-offloaded, tasks that are executed locally on the MEC and tasks executed on the cloud has been made in this section. This is shown in Figure 8. According to the figure, the maximum percentage of tasks executed locally on the MEC is 78%. The outcome displayed on the graph depends on the scheduling algorithm employed. There are some noticeable similarities and differences among the six algorithms. An increase in the number of nodes has very little effect on ALG_1 and ALG_6. However, for ALG_2 and ALG_5 (both use EDF), an increase in the number of nodes increases the number of tasks re-offloaded to MEC and cloud. This has an opposite effect on ALG_3 and ALG_4 which both use RMS for scheduling.

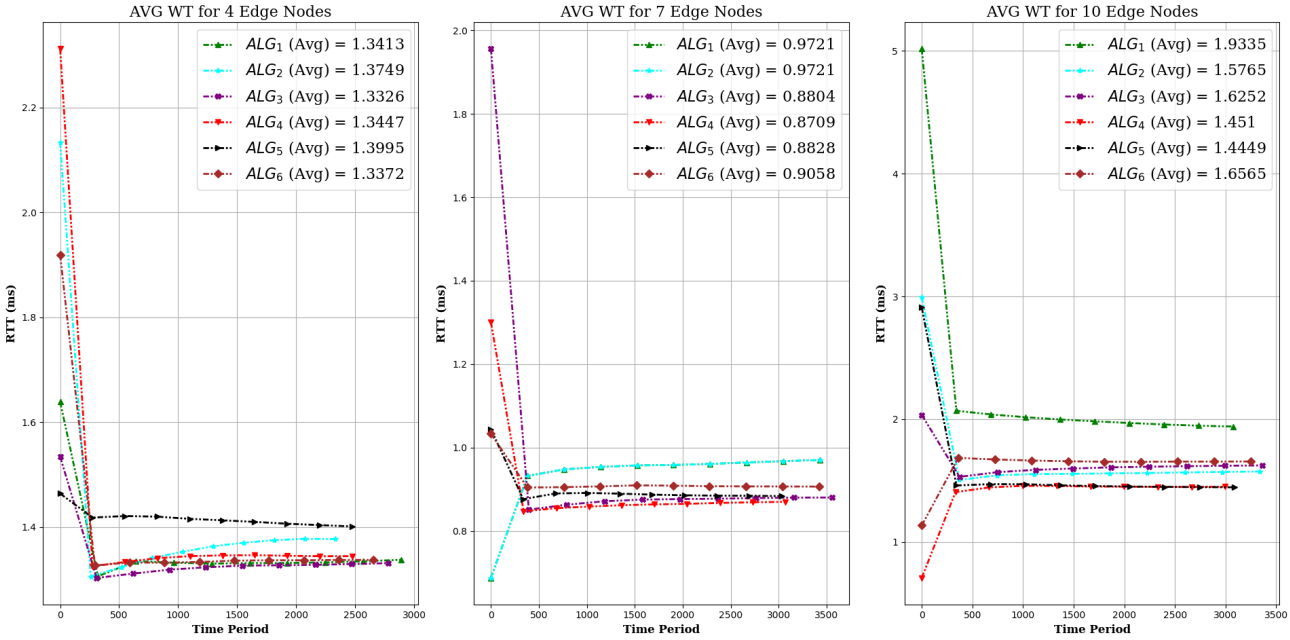

Fig. 7. The waiting time obtained for each of the algorithms during the experiments in milliseconds and the average waiting time for Exp1 is displayed.

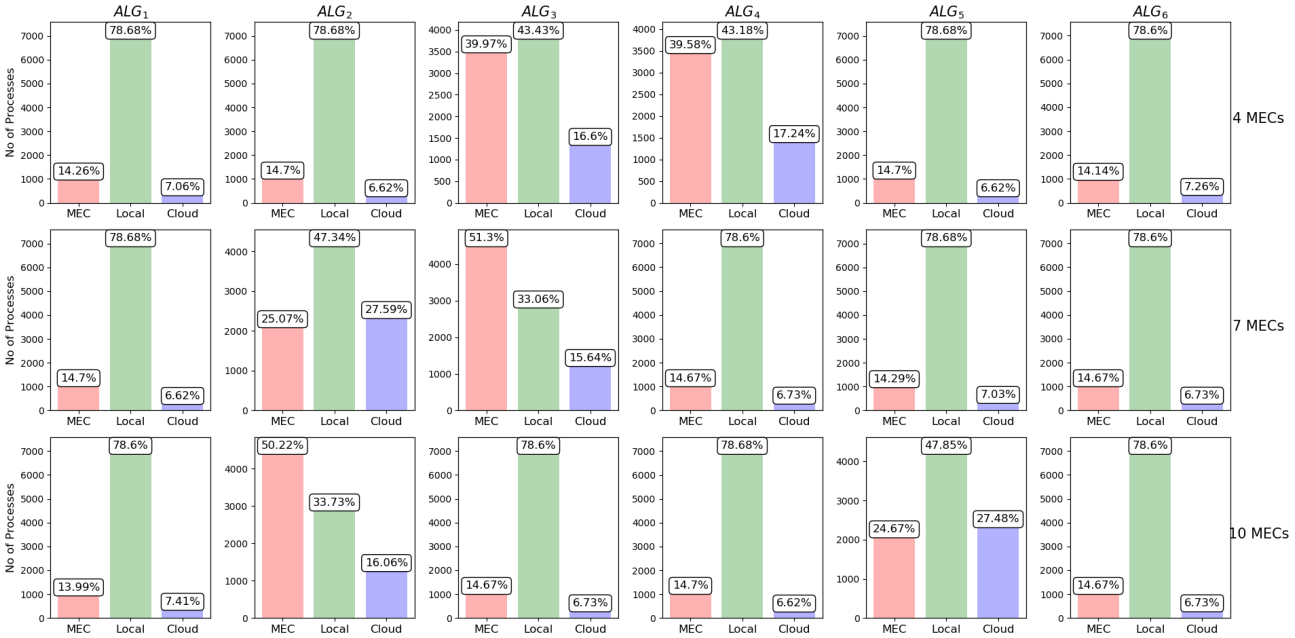

Fig. 8. The figure shows a comparison of the number of tasks that have been executed in the local MEC, re-offloaded to another MEC or cloud in Exp1

5) Comparison of the ratio of processes that meet Execution Deadline

In this section, the ratio of tasks that meet their execution deadline during the experiment for Exp_1 is compared for all six compared algorithms. These results are obtained from the end device. Each task that is sent out by the end device to the MEC node has a deadline constraint and is monitored to make sure that the deadline constraint is met as the task travels through the MEC platform and back to the end node. The percentage of tasks that meets the deadline constraint is labelled here as TP (Timely Process) while the percentage of tasks that did not meet the deadline constraint is labelled here as UP (Untimely Process). It can be seen in Figure 9 that more tasks meet the deadline as the number of MECs increases. It can also be seen that more tasks meet their deadline using ALG_4 with 10 MECs compared to the other algorithms in this Exp_1. Furthermore, ALG_6 obtains the least number of untimely processes with 4 MECs.

Fig. 9. Comparison of the ratio of processes that missed their deadline to the processes that failed to meet the deadline. TP (Timely process) is used to denote processes that meet the execution deadline while UP (Untimely Process) is used to denote tasks processes that missed the deadline for Exp1

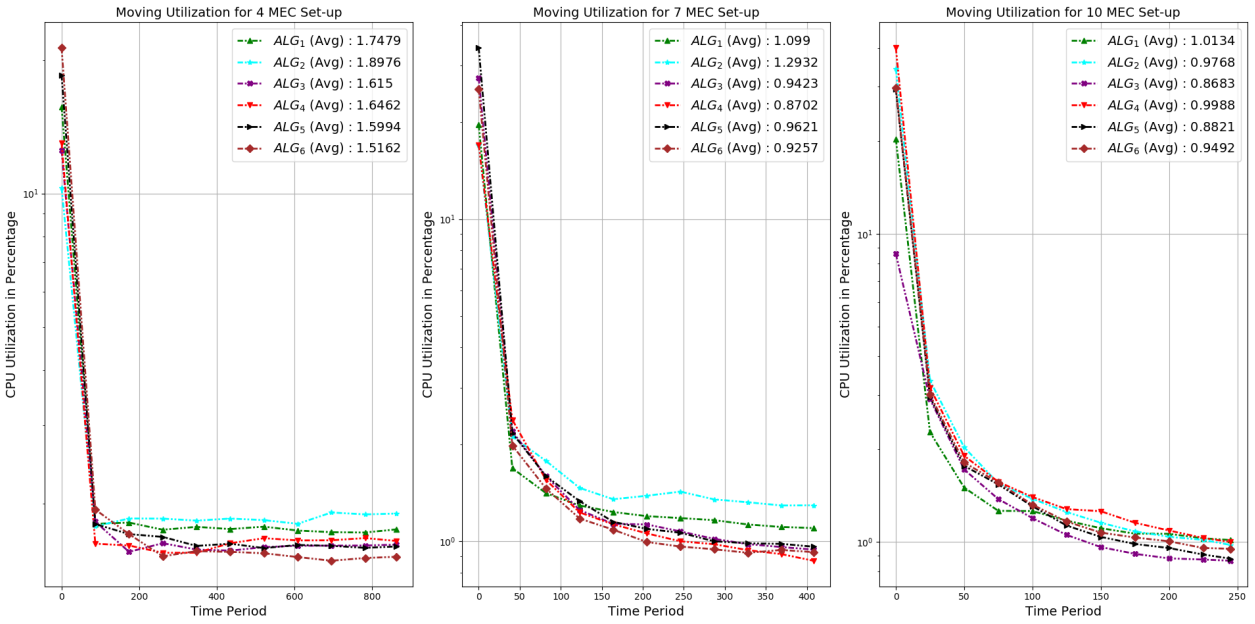

Fig. 10. The CPU utilization for each of the algorithm during the experiments in percentage and the average CPU utilization for Exp2 is displayed.

B. Experimental Results for Experimental setup2 (Exp2)

This section contains experimental results obtained during experimentation using the Exp_2 setup.

1) CPU Comparison

Figure 10 shows the CPU utilization results obtained for the algorithms during the Exp_2. As depicted in the figure, the CPU utilization for each of the compared algorithms decreases with an increase in the MEC node. This gradual decrease is similar to the behaviour in Exp_1. This can be attributed to the sharing of the total workload sent by clients among the MEC nodes. It can also be seen that the CPU utilization of the case study algorithms for Exp_2 is lower than the Exp_1. This observation is trivial as in Exp_1, the MECs were equally busy throughout the experiment which is not the case in Exp_2. Averaging the results of the 3 sub experiments with 4, 7 and 10 MECs, ALG_2 obtains the highest CPU utilization while ALG_6 achieves the slowest CPU utilization during the experiments. ALG_2 uses a deadlock avoidance algorithm while ALG_6 uses a deadlock prevention algorithm.

2) Round Trip Time

The round-trip time obtained for Exp_2 can be seen in Figure 11. Each participating MEC records the RTT for each of the

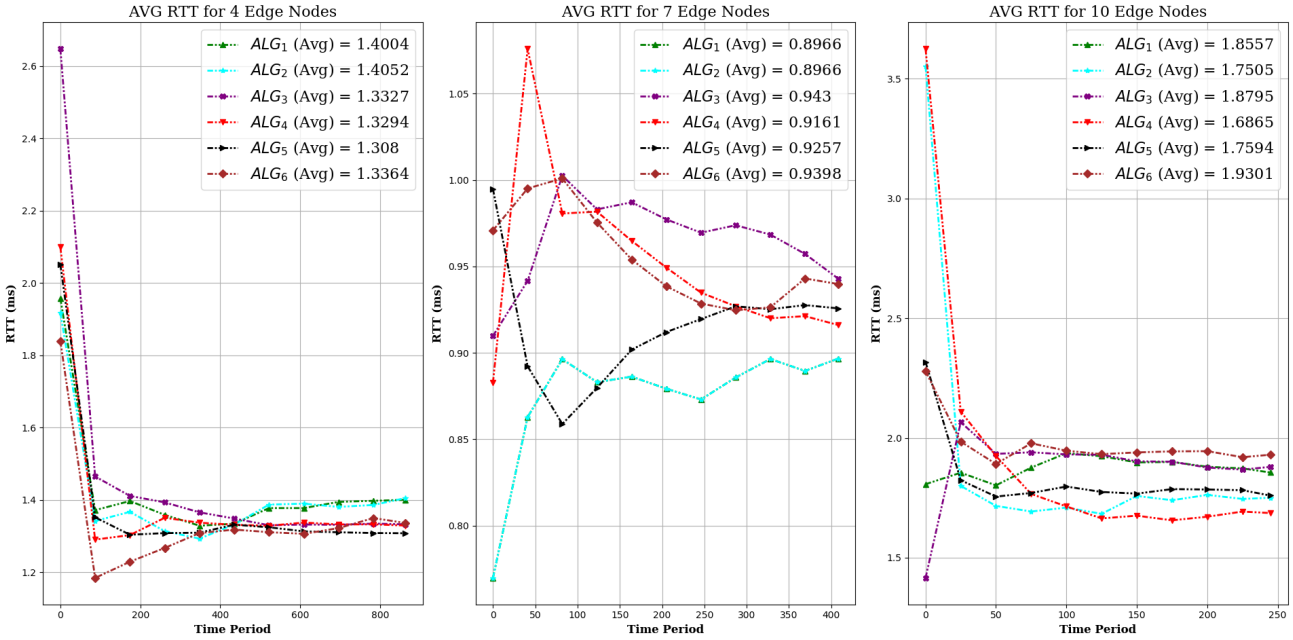

Fig. 11. The RTT obtained for each of the algorithms during the experiments in milliseconds and the average RTT for Exp2 is displayed.

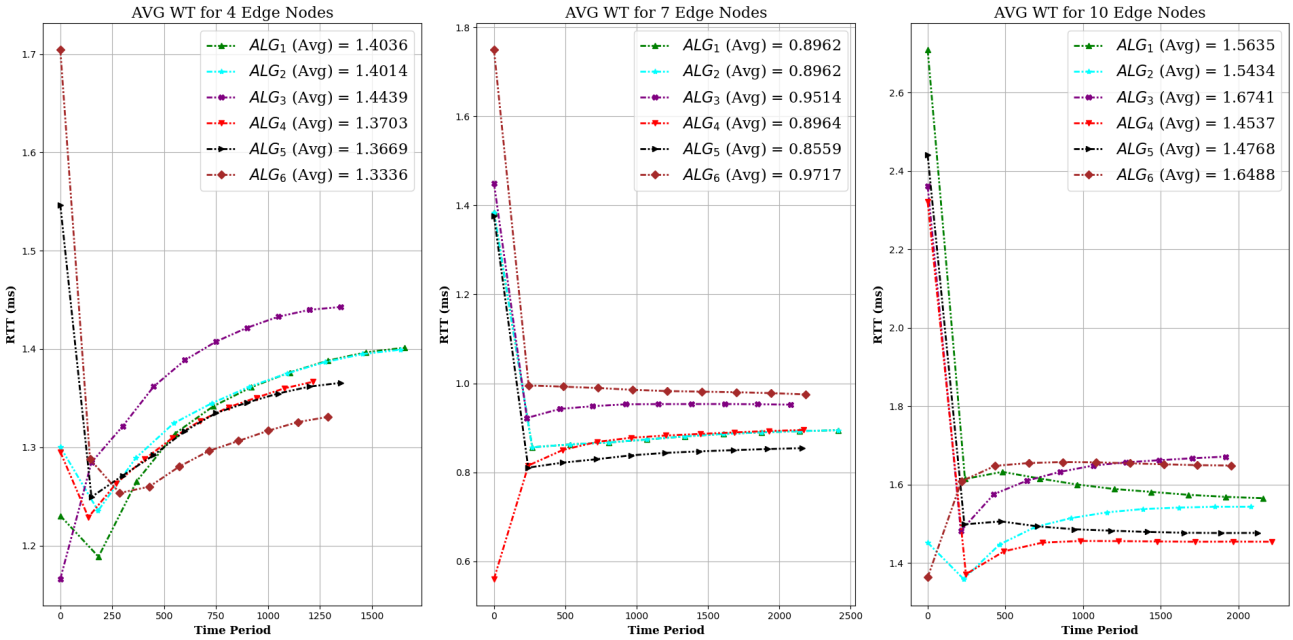

Fig. 12. The waiting time obtained for each of the algorithms during the experiments in milliseconds and the average waiting time for Exp2 is displayed.

MEC in the platform to be used for offloading decisions. The RTT is obtained here in the Exp_2 setup similar to how it is obtained in the Exp_1 setup. The round-trip time for the Exp_2 ranged between 0.89 to 1.93 milliseconds. Averaging the results of the 3 sub experiments with 4, 7 and 10 MECs, ALG_4 obtains the lowest overall RTT while ALG_6 achieves the highest RTT. ALG_4 and ALG_6 are both deadlock prevention algorithms. However, ALG_4 uses RMS for task scheduling while ALG_6 uses EDF.

3) Waiting Time

The waiting time convergence comparison result for the Exp_2 has been depicted in Figure 12. Exp_2 setup seems to have a more predictable convergence than the Exp_1 as the waiting time converges between 0.89 to 1.63 milliseconds for each of the experimental runs from 4 to 10 MECs. On average, ALG_4 attains a lower waiting time while ALG_1 acquires a higher waiting time during the 3 sub-experiments from 4 to 10 MECs. ALG_4 utilises a deadlock prevention algorithm while ALG_1 employs a deadlock avoidance algorithm.

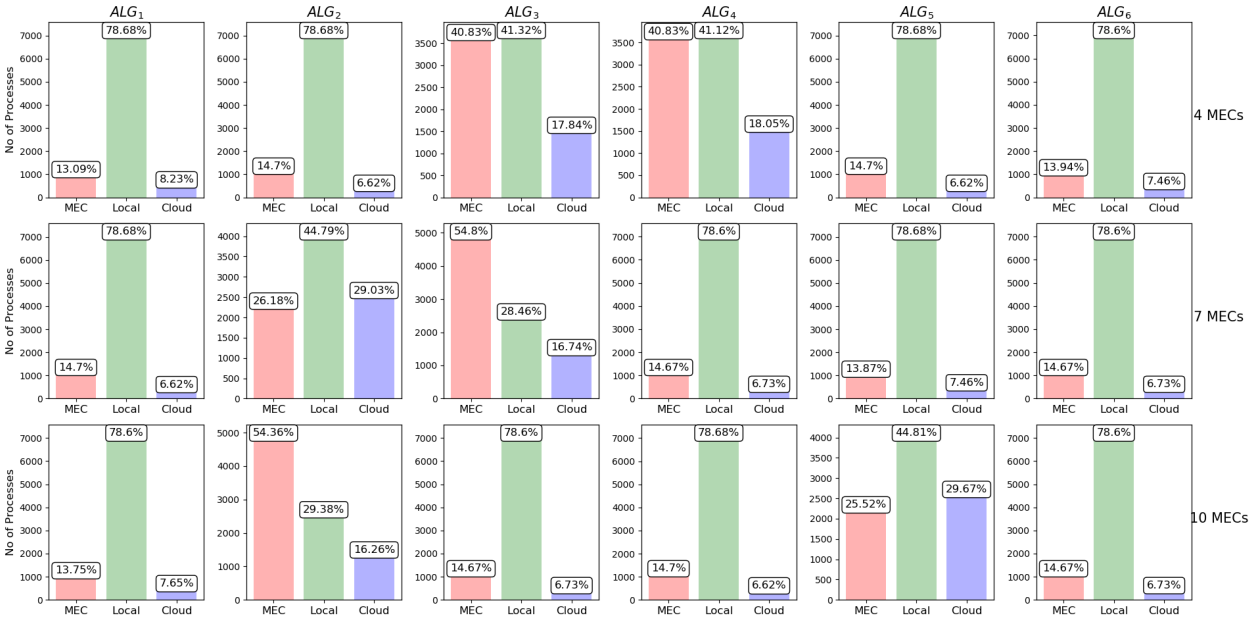

Fig. 13. Comparison Between the ratio of processes that were executed locally in the MEC to processes that were Re-offloaded for Exp2

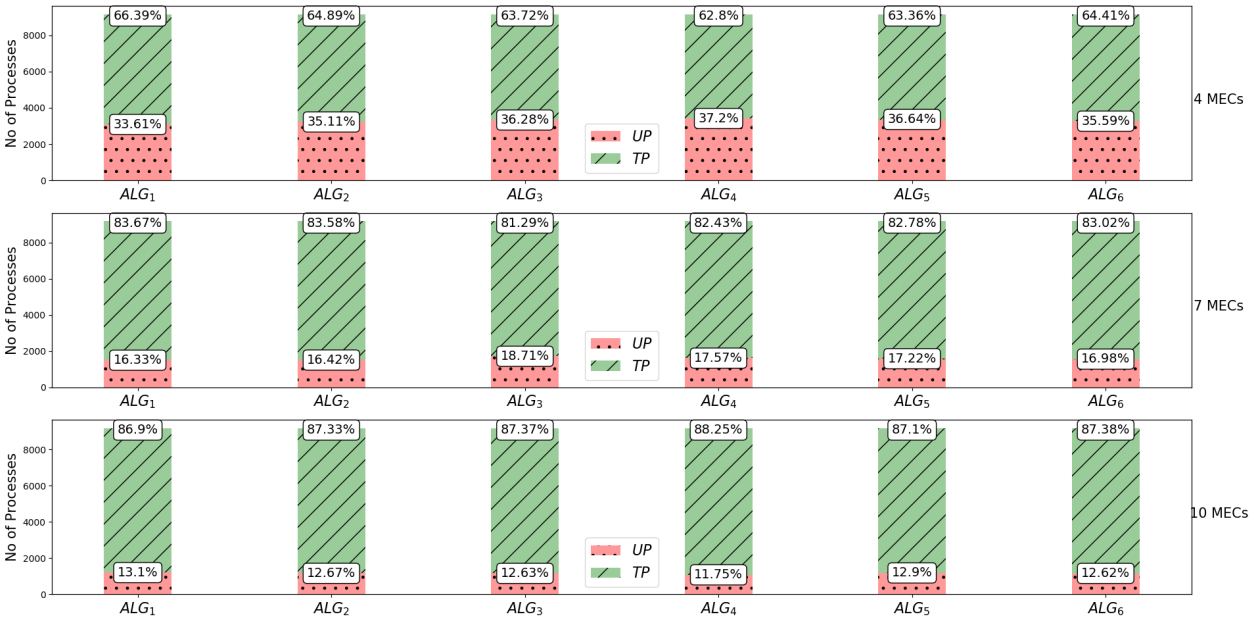

Fig. 14. Comparison of the ratio of processes that missed their deadline to the processes that failed to meet the deadline. TP (Timely process) is used to denote processes that meet the execution deadline while UP (Untimely Process) is used to denote tasks processes that missed the deadline for Exp2

4) Offload vs Local

The comparisons between the ratio of tasks executed locally, re-offloaded to the cloud or re-offloaded to another MEC are presented in this section. It can be seen in Figure 13 that an increase in the number of MECs leads to an increase in the percentage of tasks re-offloaded to a neighbouring MEC for all compared algorithms. The overall behaviour here is similar to what is shown in Exp_1. ALG_6 and ALG_1 algorithm had the best performance result with the number of tasks executed locally for each run above 78%. However, an increase in the number of nodes has very little effect on ALG_6 and ALG_1 similar as in Exp_1. ALG_2 achieves the highest increase rate in tasks re-offloaded to be executed in a neighbouring MEC as the number of MECs increases.

5) Comparison of the ratio of processes that meet Execution Deadline

In this section, the ratio of tasks that meet their execution deadline during the experiment for Exp_2 has been compared for all six algorithms. These results are obtained from the end device perspective. Each task that is sent out by the end device to the MEC node has a deadline constraint and is monitored to make sure that the deadline constraint is met as the task

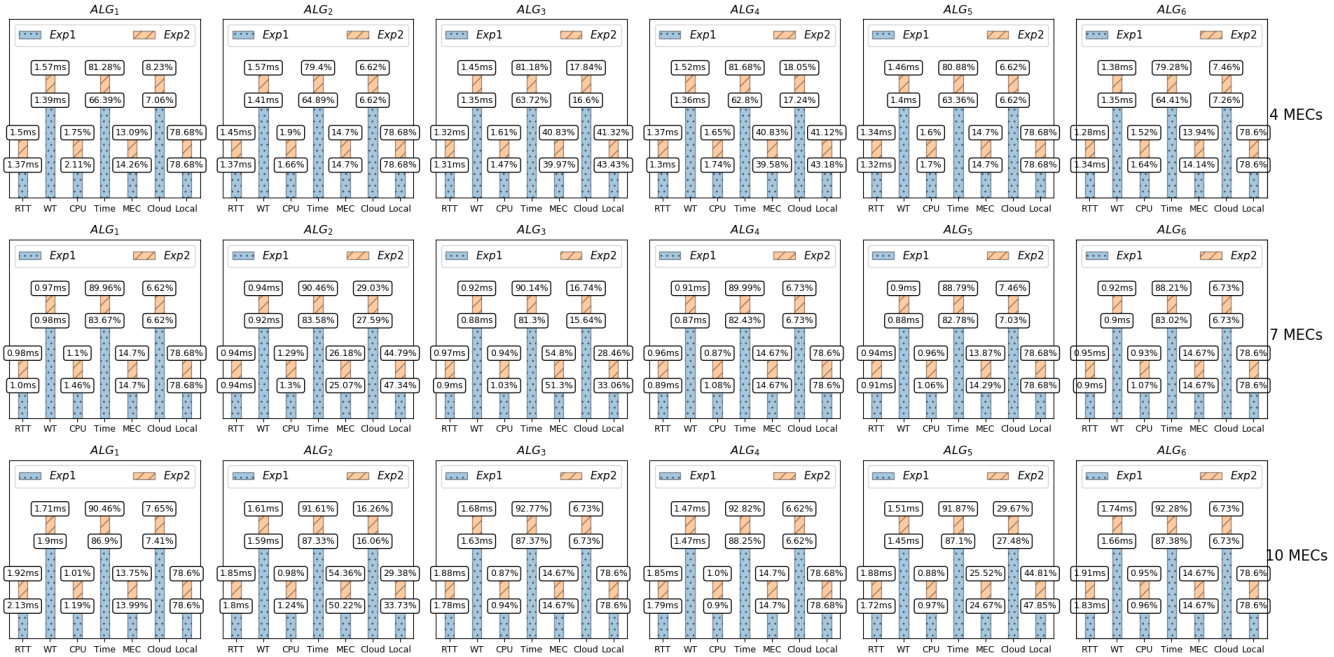

Fig. 15. Summary Comparison of the results obtained in Exp1 and Exp2 (Graph is not drawn to scale)

travels through the MEC platform and back to the end node. The percentage of tasks that meets the deadline constraint is labelled here as TP (Timely Process) while the percentage of tasks that did not meet the deadline constraint is labelled here as UP (Untimely Process). It can be seen in Figure 14 that more tasks meet the deadline as the number of MECs increases. It can also be seen that more tasks meet their deadline while using the ALG_2 for each experimental run from 4 MECs to 10 MECs compared to the other algorithms. ALG_1 obtains the lowest TP for 10 MECS while ALG_4 achieve the highest TP. ALG_4 utilises a deadlock prevention algorithm while ALG_1 employs a deadlock avoidance algorithm.

6) Exp1 and Exp2 Comparison

Figure 15 summarizes the difference in the experimental outcomes of Exp_1 and Exp_2. The figure shows the average outputs of each of the experimental runs (4, 7 and 10). The figure shows that ALG_3 obtains better CPU utilization compared to other algorithms. ALG_6 and ALG_1 obtains better percentage of tasks executed locally compared to other algorithms in Exp_1 and Exp_2. Comparing algorithms that obtain better overall percentage of tasks executed on time, ALG_4 provides the best performance among all the algorithms under study. ALG_1 and ALG_2 use a deadlock avoidance algorithm while ALG_3 uses a deadlock prevention algorithm. To generalize, avoidance algorithm does better in the percentage of tasks executed locally and the overall percentage of tasks executed on time while prevention algorithms obtain better CPU utilization. Additionally, the optimum difference between these two algorithms is the ability to keep the system in a safe state. Exp_1 and Exp_2 obtain similar behavioural patterns for the algorithms when comparing the CPU utilization and the offload vs local.

VI. Conclusion

In this paper, a comparative analysis of Deadlock Avoidance and Prevention Algorithms for Resource Provisioning in Collaborative Edge Computing has been presented for Autonomous Vehicles (AVs). The study has been carried out in a step on building reliable and readily available MEC platforms that can be used to deliver low latency requirements for 6G case study scenarios. Using a case study for real-time AV systems in this study, comparisons were made on deadlock situations in real-time systems. Rate Monotonic scheduling algorithm or Earliest Deadline First algorithm were used in prioritizing the workloads. In this research, our hypothesis was established based on our previous study on deadlocks in MEC by comparing six algorithms. Two experimental setups were designed on the GNS3 platform for evaluating the compared algorithms. The metrics used in the comparison include Round-trip time, Queue waiting time, CPU utilization and the ratio of tasks. The contributions made in this paper also have the potential to be handy in deadlock avoidance and prevention for efficient resource provisioning in AV networks. One major limitation of the research is that it has been carried out in a closed environment with limited specifications. It would be interesting to know if the results obtained are reproduced in different environments.

VII. Future Work

Future work in this area may be to explore deadlock prediction in MEC using data collected over time. In this scenario, the Resource Allocation Graph (RAG) maps the resource consumption over time for each MEC. Additionally, the fluctuation of resource consumption for each edge is monitored which results in a time series. Furthermore, an autoregressive function may be used to estimate the probability

of the RAG to form a cycle. Hence, a proactive deadlock algorithm is used to deprioritise processes that are more likely to result in a deadlock as a preventive measure.

REFERENCES

[1] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.

[2] A. Hussain, M. Iqbal, S. Sarwar, M. Safyan, Z. ul Qayyum, H. Gao, and X. Wang, "Servicing delay sensitive pervasive communication through adaptable width channelization for supporting mobile edge computing," *Computer Communications*, vol. 162, pp. 152–159, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366420318399

[3] M. Rizwan, A. Nadeem, M. Iqbal, S. Sarwar, M. Safyan, and Z. U. Qayyum, "An adaptive software fault-tolerant framework for ubiquitous vehicular technologies," *IEEE Communications Standards Magazine*, vol. 4, no. 4, pp. 26–32, 2020.

[4] C. R. Kalmanek and Y. R. Yang, "The challenges of building reliable networks and networked application services," in *Guide to Reliable Internet Services and Applications*. Springer, 2010, pp. 3–17.

[5] S. A. Hussain, M. Iqbal, A. Saeed, I. Raza, H. Raza, A. Ali, A. K. Bashir, and A. Baig, "An efficient channel access scheme for vehicular ad hoc networks," *Mobile Information Systems*, vol. 2017, 2017.

[6] S. Shahzadi, M. Iqbal, and N. R. Chaudhry, "6g vision: Toward future collaborative cognitive communication (3c) systems," *IEEE Communications Standards Magazine*, vol. 5, no. 2, pp. 60–67, 2021.

[7] M. A. Khan, S. Ghosh, S. A. Busari, K. M. S. Huq, T. Dagiuklas, S. Mumtaz, M. Iqbal, and J. Rodriguez, "Robust, resilient and reliable architecture for v2x communications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4414–4430, 2021.

[8] L. Feng, A. Ali, M. Iqbal, F. Ali, I. Raza, M. H. Siddiqi, M. Shafiq, and S. A. Hussain, "Dynamic wireless information and power transfer scheme for nano-empowered vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4088–4099, 2020.

[9] N. Kherraf, H. A. Alameddine, S. Sharafeddine, C. M. Assi, and A. Ghrayeb, "Optimized provisioning of edge computing resources with heterogeneous workload in iot networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 459–474, 2019.

[10] S. Ghosh, T. Dagiuklas, M. Iqbal, and X. Wang, "A cognitive routing framework for reliable communication in iot for industry 5.0," *IEEE Transactions on Industrial Informatics*, 2022.

[11] S. Sarwar, S. Zia, Z. ul Qayyum, M. Iqbal, M. Safyan, S. Mumtaz, R. García-Castro, and K. Kostromitin, "Context aware ontology based hybrid intelligent framework for vehicle driver categorization," *Transactions on Emerging Telecommunications Technologies,*, 2019.

[12] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts, 9th Edition*. Wiley, 2012. [Online]. Available: https://books.google.co.uk/books?id=9VMcAAAAQBAJ

[13] E. E. Ugwuanyi, S. Ghosh, M. Iqbal, and T. Dagiuklas, "Reliable resource provisioning using bankers' deadlock avoidance algorithm in mec for industrial iot," *IEEE Access*, vol. 6, pp. 43 327–43 335, 2018.

[14] E. E. Ugwuanyi, M. Iqbal, and T. Dagiuklas, "A novel predictive-collaborative-replacement (pcr) intelligent caching scheme for multi-access edge computing," *IEEE Access*, vol. 9, pp. 37 103–37 115, 2021.

[15] A. Lankes, T. Wild, A. Herkersdorf, S. Sonntag, and H. Reinig, "Comparison of deadlock recovery and avoidance mechanisms to approach message dependent deadlocks in on-chip networks," in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, 2010, pp. 17–24.

[16] V. Kate, A. Jaiswal, and A. Gehlot, "A survey on distributed deadlock and distributed algorithms to detect and resolve deadlock," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*, 2016, pp. 1–6.

[17] Y. Choi, J. Kwon, S. Jeong, H. Park, and Y. I. Eom, "Work-in-progress: Lightweight deadlock detection technique for embedded systems via os-level analysis," in *2018 International Conference on Embedded Software (EMSOFT)*, 2018, pp. 1–2.

[18] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Risk-based optimization of resource provisioning in mobile edge computing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, 2018, pp. 328–330.

[19] P. Chang and G. Miao, "Resource provision for energy-efficient mobile edge computing systems," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[20] S. Yu and R. Langar, "Collaborative computation offloading for multi-access edge computing," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. IEEE, 2019, pp. 689–694.

[21] Z. Zhou, S. Yu, W. Chen, and X. Chen, "Ce-iot: Cost-effective cloud-edge resource provisioning for heterogeneous iot applications," *IEEE Internet of Things Journal*, 2020.

[22] Y. Ma, W. Liang, J. Li, X. Jia, and S. Guo, "Mobility-aware and delay-sensitive service provisioning in mobile edge-cloud networks," *IEEE Transactions on Mobile Computing*, 2020.

[23] Y. Ma, W. Liang, M. Huang, Y. Liu, and S. Guo, "Virtual network function service provisioning for offloading tasks in mec by trading off computing and communication resource usages," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 1–7.

[24] N. Farajzadeh, M. Hashemzadeh, M. Mousakhani, and A. T. Haghighat, "An efficient generalized deadlock detection and resolution algorithm in distributed systems," in *The Fifth International Conference on Computer and Information Technology (CIT'05)*, 2005, pp. 303–309.

[25] A. Izumi, T. Dohi, and N. Kaio, "Deadlock detection scheduling for distributed processes in the presence of system failures," in *2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing*, 2010, pp. 133–140.

[26] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," in *Concurrency: the Works of Leslie Lamport*, 2019, pp. 179–196.

[27] K. M. Chandy, J. Misra, and L. M. Haas, "Distributed deadlock detection," *ACM Transactions on Computer Systems (TOCS)*, vol. 1, no. 2, pp. 144–156, 1983.

[28] S. Davidson, I. Lee, and V. F. Wolfe, "Deadlock prevention in concurrent real-time systems," *Real-Time Systems*, vol. 5, no. 4, pp. 305–318, 1993.

[29] L. Lou, F. Tang, I. You, M. Guo, Y. Shen, and L. Li, "An effective deadlock prevention mechanism for distributed transaction management," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2011, pp. 120–127.

[30] J. Ding, H. Zhu, H. Zhu, and Q. Li, "Formal modeling and verifications of deadlock prevention solutions in web service oriented system," in *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*. IEEE, 2010, pp. 335–343.

[31] Z. Chuanfu, L. Yunsheng, Z. Tong, Z. Yabing, and H. Kedi, "A deadlock prevention approach based on atomic transaction for resource co-allocation," in *2005 First International Conference on Semantics, Knowledge and Grid*. IEEE, 2005, pp. 37–37.

[32] W. M. van der Aalst, "Orchestration," in *Encyclopedia of Database Systems*. Springer, 2009, pp. 2004–2005.

[33] S.-D. Lang, "An extended banker's algorithm for deadlock avoidance," *IEEE Transactions on software engineering*, vol. 25, no. 3, pp. 428–432, 1999.

[34] F. Tricas, J. M. Colom, and J. Ezpeleta, "Some improvements to the banker's algorithm based on the process structure," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 3. IEEE, 2000, pp. 2853–2858.

[35] J. J. Lee and V. J. Mooney, "A novel {O (n)} parallel banker's algorithm for system-on-a-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 12, pp. 1377–1389, 2006.

[36] P. M. El-Kafrawy, "Graphical deadlock avoidance," in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*. IEEE, 2009, pp. 308–312.

[37] P. Li, K. Agrawal, J. Buhler, R. D. Chamberlain, and J. M. Lancaster, "Deadlock-avoidance for streaming applications with split-join structure: Two case studies," in *ASAP 2010-21st IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2010, pp. 333–336.

[38] A. Mohammadi and S. G. Akl, "Scheduling algorithms for real-time systems," *School of Computing Queens University, Tech. Rep*, 2005.

[39] T. H. Team, "Quality of Service 0,1 & 2 - MQTT Essentials: Part 6," library Catalog: www.hivemq.com. [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels

[40] Andrew Coleman, Julien Duponchelle, and Ricar Ganancial, "GNS3 documentation," 2019. [Online]. Available: https://docs.gns3.com/

[41] "The CLOUDS Lab: Flagship Projects - Gridbus and Cloudbus." [Online]. Available: http://www.cloudbus.org/cloudsim/

[42] nsnam, "ns-3." [Online]. Available: https://www.nsnam.org/documentation/

[43] E. E. Ugwuanyi, "deadlock project," Dec. 2019, original-date: 2019-05-06T13:59:31Z. [Online]. Available: https://github.com/emylincon/deadlock_project