ISSN: 2229-6948(ONLINE)
DOI: 10.21917/ijct.2024.0480

AN ENSEMBLE ADAPTIVE REINFORCEMENT LEARNING BASED EFFICIENT LOAD BALANCING IN MOBILE AD HOC NETWORKS

G. Rajiv Suresh Kumar[1] and G. Arul Geetha[2]
[1]*Department of Information Technology, Hindusthan College of Engineering and Technology, India*
[2]*Department of Computer Science, Bishop Appasamy College of Arts and Science, India*

Abstract

This research work introduces an Ensemble Adaptive Reinforcement Learning (EARL) approach for efficient load balancing in Mobile Ad Hoc Networks (MANETs). Traditional methods often fail to adapt to the dynamic nature of MANETs, leading to congestion and inefficiency. EARL leverages multiple reinforcement learning agents, trained with Q-learning and Deep Q-Networks (DQN), to optimize routing decisions based on real-time network conditions. The ensemble mechanism combines the strengths of individual agents, enhancing adaptability and performance. Simulation results demonstrate that EARL significantly outperforms traditional methods like AODV and DSR, achieving higher packet delivery ratios, lower end-to-end delays, increased throughput, better energy efficiency, and reduced packet loss, thereby proving its effectiveness in dynamic network environments.

Keywords:

Ad Hoc Networks, Load Balancing, Adaptive, Learning, Efficient

1. INTRODUCTION

Mobile ad hoc networks, also known as MANETs, have rapidly become a popular means of networking since they are easy to use, have a low-cost structure, and can perform well over greater areas. Due to the absence of a major control node, the networks have the potential to become highly slow and inefficient. This circumstance presents a challenge for resource allocation algorithms, even though the comfort level has grown [1]. There are a variety of research companies that have provided excellent resource allocation systems that employ a wide variety of strategies to address this problem. Within the context of cloud computing, a wide range of approaches to the distribution of MANET resources are presented as potential options [2].

Methods that utilize cloud computing have the potential to create better overall community performance than other techniques. This is since cloud computing can maximize resources across a huge region with less effect from visitor congestion. Postpone throughput, dependability, and efficiency are the three parameters that can be utilized to measure the effectiveness of these algorithms [3].

When it comes to the distribution of assistance, a solid set of guidelines ought to perform well in all three of these categories. Since it has an immediate impact on the entire reaction time of the consumer, delay is an essential statistic that should be considered when evaluating the effectiveness of a set of rules for allocation inevitable that latency will grow as extra resources are allocated because of the overhead that is necessary to process requests. Throughput is an important attribute because efficient resource allocation algorithms need to be able to handle applications that are very sensitive to networks with limited bandwidth. This is because throughput is a measure of communication speed. The reliability of an allocation algorithm is therefore an essential component in establishing the effectiveness of the method [4].

Customers will not find much use for a method for aid allocation if it is unable to provide trustworthy overall performance in situations when there is a considerable amount of competition. The process may get even more complicated if cloud algorithms are incorporated into MANET resource allocation algorithm evaluations. This is because the overall performance of the algorithms will now need to be examined within the standard system. The algorithms that are stored in the cloud could optimize assets across a vast area; nevertheless, there are certain circumstances in which they will not deliver ideal results, which will result in a decrease in performance. To determining how efficient cloud computing algorithms are at distributing resources, it is necessary to conduct simulations in order to see the outcomes of various situations that occur in the real world [5].

Researchers can make use of these simulations in order to examine the robustness of the algorithms under a variety of scenarios, which ultimately assists them in developing more effective allocation methods. The methodologies that are typically used in cloud computing offer an up-to-date method for optimizing assets in MANETs. A proper evaluation of these algorithms and the development of more robust aid allocation mechanisms requires an efficient evaluation approach that simulates the conditions of real-world networks. This is important to ensure that the evaluation is accurate. Cellular ad hoc networks, also known as MANETs, are utilized by a multitude of military and civilian applications to deliver a variety of services. These services include disaster aid, vehicle programs, and rapid network deployment. These services are provided without the necessity of establishing permanent communication links [6].

In line with the development of MANETs, there has been an increase in the desire for resource allocation strategies that are less harmful to the environment. The dilemma of how to most effectively distribute resources in these kinds of networks can be solved in a realistic manner by using algorithms that are performed on the cloud. By examining records and employing sophisticated modeling approaches, algorithms in the cloud can extract insight from complex datasets. This is accomplished to determine the most effective ways to distribute resources.

In MANETs, it is feasible to make more efficient use of the resources that are available by combining these concepts with operational strategies for the distribution of those resources. Methods of cloud computing have the potential to improve communication latency, congestion control, and strength intake, in addition to achieving the most effective utilization of valuable assistance. Algorithms in the cloud can, as they should, sense the quality solutions for certain network scenarios. This is in addition to ensuring that assets are used appropriately. MANETs present a difficult challenge when it comes to resource allocation because

of the many different spoken communication settings and the mobility of the nodes [7].

Inefficiency, low scalability, and dynamic source switching are only some of the issues that plague traditional solutions despite their many advantages [8]. Cloud computing methods have the potential to provide more effective and dependable asset control in MANETs. This is accomplished by enabling users to make decisions based on real-time data evaluation and improved modeling methodologies. In addition, the techniques for allocating help in the cloud have the potential to lessen the likelihood of resource over-allocation, which can result in congestion or other problems related to intelligence [9]. Through the utilization of algorithms hosted in the cloud, administrators of cell communities can rapidly adjust to new situations by gaining an understanding of which resource allocation strategies are most effective for their MANETs.

2. PROPOSED METHOD

The proposed method, named Ensemble Adaptive Reinforcement Learning (EARL), aims to enhance load balancing efficiency in Mobile Ad Hoc Networks (MANETs). This method integrates multiple reinforcement learning (RL) agents, each adapted to different network conditions, and combines their outputs to make informed routing decisions. The core idea is to leverage the strengths of various RL agents to handle the dynamic and unpredictable nature of MANETs. EARL uses Q-learning and Deep Q-Networks (DQN) to adaptively distribute the network load, reducing congestion and improving overall network performance. Each RL agent learns optimal policies for load balancing based on real-time feedback from the network environment, and an ensemble decision mechanism aggregates these policies to select the best action for routing data packets.

2.1 ENSEMBLE ADAPTIVE REINFORCEMENT LEARNING BASED LOAD BALANCING

The proposed Ensemble Adaptive Reinforcement Learning (EARL) method for load balancing in Mobile Ad Hoc Networks (MANETs) is designed to dynamically manage network traffic and improve overall performance. MANETs are characterized by their highly dynamic and unpredictable nature due to node mobility and varying network conditions. Traditional load balancing methods often struggle to adapt to these changing conditions, leading to congestion, packet loss, and inefficient resource utilization.

EARL addresses these challenges by leveraging the strengths of multiple reinforcement learning (RL) agents. Each RL agent is trained using different RL algorithms, such as Q-learning and Deep Q-Networks (DQN), to optimize routing decisions based on specific network conditions. By training multiple agents on varied scenarios, EARL creates an ensemble of agents, each capable of handling different aspects of the network's dynamics.

In operation, each RL agent continuously learns and updates its policy based on real-time feedback from the network. The agents monitor key network metrics, such as node load, link quality, and mobility patterns, to make informed routing decisions. The ensemble approach aggregates the outputs of these agents, combining their learned policies to select the best possible route for data packets. This ensemble decision mechanism ensures that the most suitable agent's policy is applied to current network conditions, leading to improved load balancing and network performance.

EARL's adaptive nature allows it to dynamically adjust to the network's state, distributing the load more evenly across nodes and avoiding congestion. This results in higher packet delivery ratios, lower end-to-end delays, and increased throughput. Additionally, EARL's energy-efficient routing decisions help extend the battery life of network nodes, which is crucial for the sustainability of MANETs.

2.1.1 Pseudocode: EARL:

```
class RLAgent:
    def __init__(self, agent_id, algorithm):
        self.agent_id = agent_id
        self.algorithm = algorithm
        self.q_table = {}  # Q-table for Q-learning
        self.dqn_model = None  # Model for DQN if applicable
        self.epsilon = 0.1  # Exploration rate
        self.learning_rate = 0.01  # Learning rate
        self.discount_factor = 0.9  # Discount factor
    def choose_action(self, state):
        if random.random() < self.epsilon:
            return random_action()  # Exploration
        else:
            return self.best_action(state)  # Exploitation
    def best_action(self, state):
        if self.algorithm == "Q-learning":
            return max(self.q_table[state], key=self.q_table[state].get)
        elif self.algorithm == "DQN":
            return self.dqn_model.predict(state)
        else:
            return random_action()
    def update(self, state, action, reward, next_state):
        if self.algorithm == "Q-learning":
            self.q_table[state][action] = (1 - self.learning_rate) * self.q_table[state][action] + \
                                          self.learning_rate * (reward + self.discount_factor * max(self.q_table[next_state].values()))
        elif self.algorithm == "DQN":
            self.dqn_model.train(state, action, reward, next_state)
def ensemble_decision(agents, state):
    actions = [agent.choose_action(state) for agent in agents]
    best_action = select_best_action(actions)  # Function to aggregate and select the best action
    return best_action
# Simulation setup
agents = [RLAgent(agent_id=1, algorithm="Q-learning"),
          RLAgent(agent_id=2, algorithm="DQN"),
          RLAgent(agent_id=3, algorithm="Q-learning"),
```

```
        RLAgent(agent_id=4, algorithm="DQN"),
        RLAgent(agent_id=5, algorithm="Q-learning")]
# Simulation loop
while network_is_active:
    for node in network_nodes:
        state = node.get_state()
        best_action = ensemble_decision(agents, state)
        reward, next_state = node.perform_action(best_action)
        for agent in agents:
            agent.update(state, best_action, reward, next_state)
# Function to select the best action from the ensemble
def select_best_action(actions):
    action_counts = Counter(actions)
    return action_counts.most_common(1)[0][0]
```

2.1.2 Algorithm: Load Balancing:

1) **Initialization**:
   a) Initialize multiple RL agents with different RL algorithms (e.g., Q-learning, DQN).
   b) Set initial parameters for each agent, including Q-tables, DQN models, exploration rates, learning rates, and discount factors.
   c) Deploy network nodes in the simulation environment with specified parameters (e.g., node mobility, transmission range, traffic type).
2) **Network Monitoring**:
   a) Continuously monitor the network state, including metrics like node load, link quality, and mobility patterns.
   b) Each node periodically collects state information and communicates it to the RL agents.
3) **Action Selection**:
   a) For each node, use the ensemble decision mechanism to select the best action for routing data packets.
   b) Each RL agent evaluates the current state and proposes an action based on its learned policy.
   c) Aggregate the proposed actions from all agents and select the most suitable action using a predefined selection strategy (e.g., majority voting, weighted average).
4) **Data Transmission**:
   a) Nodes perform the selected routing actions to transmit data packets.
   b) Record the outcomes of these actions, including rewards (e.g., successful packet delivery, reduced latency) and the next state of the network.
5) **Learning and Update**:
   a) Update each RL agent's knowledge based on the observed rewards and the next state.
   b) For Q-learning agents, update the Q-values using the Q-learning update rule.
   c) For DQN agents, train the DQN models using the observed state-action-reward-next state tuples.

2.1.3 Pseudocode: Load Balancing:

```
class RLAgent:
    def __init__(self, agent_id, algorithm):
        self.agent_id = agent_id
        self.algorithm = algorithm
        self.q_table = {}
        self.dqn_model = None
        self.epsilon = 0.1
        self.learning_rate = 0.01
        self.discount_factor = 0.9
    def choose_action(self, state):
        if random.random() < self.epsilon:
            return random_action()
        else:
            return self.best_action(state)
    def best_action(self, state):
        if self.algorithm == "Q-learning":
            return max(self.q_table[state], key=self.q_table[state].get)
        elif self.algorithm == "DQN":
            return self.dqn_model.predict(state)
        else:
            return random_action()
    def update(self, state, action, reward, next_state):
        if self.algorithm == "Q-learning":
            self.q_table[state][action] = (1 - self.learning_rate) * self.q_table[state][action] + \
                                          self.learning_rate * (reward + self.discount_factor * max(self.q_table[next_state].values()))
        elif self.algorithm == "DQN":
            self.dqn_model.train(state, action, reward, next_state)
def ensemble_decision(agents, state):
    actions = [agent.choose_action(state) for agent in agents]
    best_action = select_best_action(actions)
    return best_action
def select_best_action(actions):
    action_counts = Counter(actions)
    return action_counts.most_common(1)[0][0]
# Simulation setup
agents = [RLAgent(agent_id=1, algorithm="Q-learning"),
          RLAgent(agent_id=2, algorithm="DQN"),
          RLAgent(agent_id=3, algorithm="Q-learning"),
          RLAgent(agent_id=4, algorithm="DQN"),
          RLAgent(agent_id=5, algorithm="Q-learning")]
# Simulation loop
while network_is_active:
    for node in network_nodes:
        state = node.get_state()
```

```
best_action = ensemble_decision(agents, state)
reward, next_state = node.perform_action(best_action)
for agent in agents:
agent.update(state, best_action, reward, next_state)
```

3. RESULTS AND DISCUSSION

The experimental setup involves a simulation environment created using the Network Simulator 3 (NS-3). The simulation is run on a cluster of high-performance computers equipped with Intel I7 processors and 128 GB of RAM. The network topology consists of 50 mobile nodes distributed randomly in a 1000m x 1000m area. Nodes move according to the Random Waypoint Mobility Model with a maximum speed of 10 m/s. Performance metrics such as packet delivery ratio (PDR), end-to-end delay, and throughput are used to evaluate the proposed method. EARL's performance is compared with traditional load balancing methods like Ad hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) protocols. Results show that EARL outperforms these traditional methods in terms of PDR, delay, and throughput, demonstrating its effectiveness in managing network load in dynamic MANET environments.

Table.1. Settings and Parameters

Parameter	Value
Simulation Tool	NS-3
Network Area	1000m x 1000m
Number of Nodes	50
Mobility Model	Random Waypoint
Maximum Node Speed	10 m/s
Transmission Range	250 meters
Traffic Type	Constant Bit Rate (CBR)
Packet Size	512 bytes
Simulation Duration	900 seconds
Number of RL Agents	5
RL Algorithms	Q-learning, Deep Q-Networks
Hardware Used	Intel I7 processors, 128 GB RAM

3.1 PERFORMANCE METRICS

- **Packet Delivery Ratio (PDR)**: This metric measures the ratio of the number of packets successfully delivered to the destination to the total number of packets sent by the source. It indicates the reliability and effectiveness of the routing protocol in ensuring data packet delivery despite network dynamics and potential link failures.
- **End-to-End Delay**: This metric calculates the average time taken for a data packet to travel from the source to the destination. It includes all possible delays caused by route discovery, queuing at the interface queue, retransmissions at the MAC layer, and propagation and transfer times. Lower end-to-end delay signifies a more efficient routing protocol in delivering data quickly across the network.
- **Throughput**: Throughput is the rate at which data packets are successfully delivered over the network, typically measured in bits per second (bps). Higher throughput indicates better network performance and efficiency in handling and transmitting large volumes of data.

Table.2. Performance Metrics

Number of Nodes	Method	Packet Delivery Ratio (PDR)	End-to-End Delay (ms)	Throughput (Mbps)
2500	AODV	85.3%	120	1.5
	DSR	82.7%	130	1.4
	EARL	91.5%	100	1.8
5000	AODV	83.2%	150	1.4
	DSR	80.5%	160	1.3
	EARL	90.2%	130	1.7
7500	AODV	81.0%	170	1.3
	DSR	78.9%	180	1.2
	EARL	88.7%	140	1.6
10000	AODV	79.5%	200	1.2
	DSR	77.0%	210	1.1
	EARL	87.3%	160	1.5

Table.3. Energy Metrics

Number of Nodes	Method	Energy Efficiency (J/bit)	Residual Energy (J)	Packet Loss (%)
2500	AODV	0.006	4500	14.7
	DSR	0.007	4600	17.3
	EARL	0.004	4700	8.5
5000	AODV	0.008	9000	16.8
	DSR	0.009	9200	19.5
	EARL	0.005	9400	9.8
7500	AODV	0.009	13500	19.0
	DSR	0.010	13800	21.1
	EARL	0.006	14200	11.3
10000	AODV	0.010	18000	20.5
	DSR	0.011	18400	23.0
	EARL	0.007	18800	12.7

- **Energy Efficiency:** This metric measures the amount of energy consumed per bit of data transmitted, with lower values indicating better energy efficiency. The proposed EARL method consistently demonstrates lower energy consumption compared to AODV and DSR. This is due to EARL's ability to adaptively manage network load and reduce unnecessary retransmissions, leading to more efficient energy usage.
- **Residual Energy:** Residual energy refers to the remaining energy in the network nodes at the end of the simulation. Higher residual energy indicates that the routing protocol is more energy-efficient and conserves battery life. EARL shows higher residual energy across all node densities,

highlighting its effectiveness in conserving energy compared to AODV and DSR.

- **Packet Loss:** Packet loss measures the percentage of data packets that fail to reach their destination. Lower packet loss indicates a more reliable and efficient routing protocol. EARL significantly reduces packet loss compared to AODV and DSR, benefiting from its adaptive load balancing and real-time route optimization.
- **Packet Delivery Ratio (PDR):** EARL outperforms AODV and DSR in PDR, demonstrating its superior ability to deliver packets reliably. This improvement is due to EARL's adaptive reinforcement learning approach, which dynamically adjusts to network conditions and reduces congestion.
- **End-to-End Delay:** EARL consistently achieves lower E2E Delay, indicating faster data transmission from source to destination. This is a result of its efficient route selection and reduced queuing delays.
- **Throughput:** EARL provides higher throughput, showing its capability to handle large volumes of data efficiently. This improvement is attributed to its effective load balancing and reduced packet loss, which ensure continuous and stable data transmission.

4. CONCLUSION

The proposed EARL method demonstrates significant improvements in load balancing for MANETs by dynamically adapting to varying network conditions. The ensemble approach, which integrates multiple RL agents, leverages the strengths of different algorithms to optimize routing decisions. This results in higher packet delivery ratios, lower end-to-end delays, increased throughput, and better energy efficiency compared to traditional methods like AODV and DSR. Additionally, EARL ability to reduce packet loss further enhances network reliability. EARL's adaptive nature allows it to effectively manage dynamic network conditions, leading to superior performance metrics. The method significantly conserves energy, extending the operational lifespan of network nodes. EARL performs well even as network size increases, demonstrating its scalability and robustness. The improvements in reliability and efficiency make EARL a viable solution for real-world MANET applications, such as disaster recovery and military communications.

REFERENCES

[1] C. Yang, J. Li, M. Guizani and M. Elkashlan "Advanced Bandwidth Sharing in 5G Cognitive Heterogeneous Networks", IEEE Wireless Communications, Vol. 15, No. 2, pp. 94-101, 2016.

[2] I. Khan, M.H. Zafar, M.T. Jan, J Lloret, M Basheri and D Singh, "Spectral and Energy Efficient Low-Overhead Uplink and Downlink Channel Estimation for 5G Massive MIMO Systems", *Entropy*, Vol. 20, No. 2, pp. 92-108, 2018.

[3] E. Hossain and V.K. Bhargava, "*Cognitive Wireless Communication Networks*", Springer Publisher, 2007.

[4] K. Dasgupta, B. Mandal, P. Dutta and J.K. Mandal, "A Genetic Algorithm (GA) based Load Balancing Strategy for Cloud Computing", *Procedia Technology*, Vol. 10, pp. 340-347, 2013.

[5] F. McLoughlin, A. Duffy and M. Conlon, "A Clustering Approach to Domestic Electricity Load Profile Characterisation using Smart Metering Data", *Applied Energy*, Vol. 141, pp. 190-199, 2015.

[6] D. Thazhathethil, N. Katre, J. Mane Deshmukh and M. Kshirsagar, "A Model for Load Balancing by Partitioning the Public Cloud", *International Journal of Innovative Research in Computer and Communication Engineering*, Vol. 2, No. 1, pp. 2466-2471, 2014.

[7] Debraj De and Sajal K. Das, "SREE-Tree: SelfReorganizing Energy-Efficient Tree Topology Management in Sensor Networks", *Proceedings of International Conference on Sustainable Internet and ICT for Sustainability*, pp. 113-119, 2015.

[8] Antoni Morell et al., "Data Aggregation and Principal Component Analysis in WSNs", IEEE Transactions on Wireless Communications, Vol. 15, No. 6, pp. 3908-3919, 2015.

[9] D.S.K. Tiruvakadu and V. Pallapa, "Confirmation of Wormhole Attack in MANETs using Honeypot", *Computers and Security*, Vol. 76, No. 2, pp. 32-49, 2018.