SAPIENZA
UNIVERSITÀ DI ROMA

Autonomous Networking a.y. 22-23

Homework 2: Report

Alessio Lucciola - lucciola.1823638@studenti.uniroma1.it
Danilo Corsi - corsi.1742375@studenti.uniroma1.it
Domiziano Scarcelli - scarcelli.1872664@studenti.uniroma1.it

January 11, 2023

Contents

1 Introduction

In this homework, we will implement a **Q-Learning-based Stepwise Routing Protocol** for multi-UAV Networks based on the paper that was presented to us. In the first part of this report, we are going to describe the **stepwise node discovery** operation, from sending and receiving packets to updating the hop count and node tables of each drone. We'll also show the flowchart of this operation and describe a way to avoid cycles inside of the network. In the second part, we are going to describe how we implemented the Q-Learning algorithm, how we structured the **relay selection**, **reward function**, and **link stability** for each drone. In the third part, we will go on to describe the changes we made to the simulator and then discuss the results obtained from the tests performed, comparing them with baseline approaches (*geographic routing*, *random routing*, and our HW1 Q-Learning solution). We'll also see how can we modify the proposed algorithm by adding geographical information to obtain better overall performances. Finally, we collected all the observations and made a summary of all the results.

2 Discovery

The **Q-Learning Based Stepwise Routing Protocol** that we tried to implement considers the minimum number of hops to the destination and link stability that is computed through a simple stepwise node discovery. The multi-UAV network structure proposed in the paper assumes to have one control center (CC) that we assume to be the depot and a number of UAVs. The discovery process is initiated by the depot and performed at every step in order to capture any changes in the network topology. The discovery phase's purpose is for the depot to know the overall information about the network, meaning all the nodes that are reachable multi-hop or single-hop with all their info. The depot stores these pieces of information in a data structure called **NodeTable**. A piece of information about a single drone has the following structure:

- *id*: the identification number of the drone;
- *hop*: number of hops that are necessary for a packet to be sent from the depot to the drone;
- $droneVelocity$: the current velocity of the drone;
- *location*: the drone's current coordinates.

Each drone stores the same pieces of information in another data structure called **NeighborTable**. In this case, the drones inside this table are only the nodes in the neighborhood, meaning the drones that are in its communication range.

Note that in this report we will indicate the father-child relationship between two nodes with the following diagram, where an arrow from node 2 to node 1 means that 1 has received a packet from 2, and 2 is the parent node of 1.

Figure 1: Node - parent node relationship

2.1 Start of the discovery

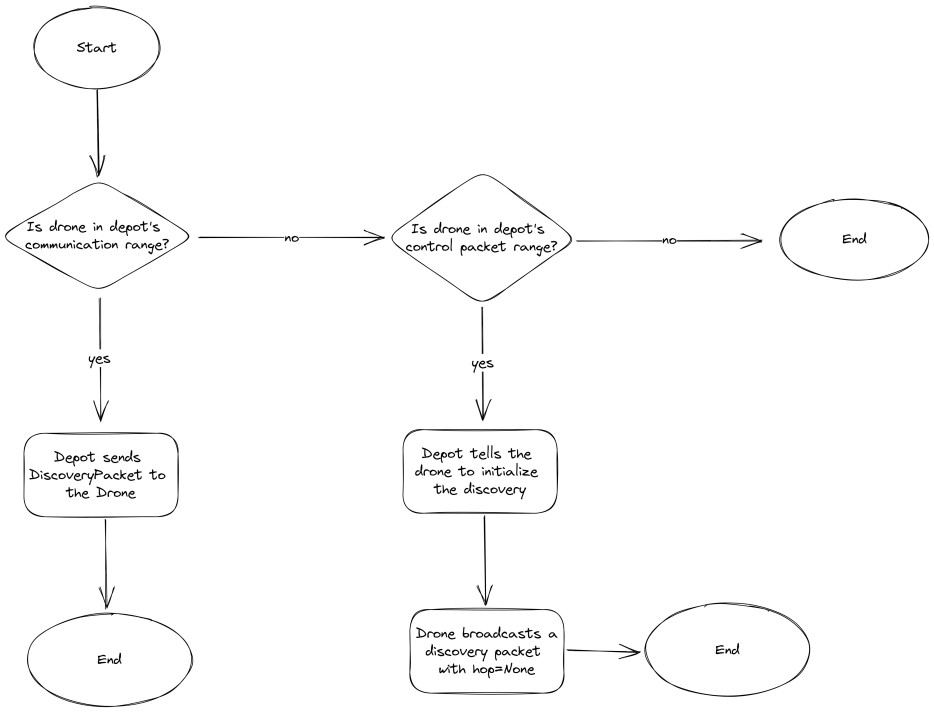

Figure 2: Depot's discovery initialization

The discovery phase is initialized by the Depot. It first sends a discovery packet to all the drones that are in its communication range. The discovery packet embeds the following information:

- Sender
- Hop count from the depot

Each drone also keeps the information about its distance from the depot, measured in the number of hops. Every drone initializes this value to None, meaning it's not defined. The depot, when creating the discovery packet, sets the initial hop count to 0. Since some drones may be disconnected from the depot, meaning there isn't a chain of drones from which a packet can travel in order to arrive at the depot. Since they need to route some packets anyways, the depot can send a long-range control packet to all the drones that are in this long communication range, in order to tell those disconnected drones to initialize the discovery phase on their own. In this case, the discovery packet that is sent from the drone that receives the long-range signal from the depot has an initial hop count equal to None. This is because the notion of "hop count from the depot" isn't defined, since there isn't a way for a packet to arrive at the depot.

2.2 Reception of a packet

A drone can receive three types of packets:

- Discovery packet
- Ack packet
- Neighbor info packet

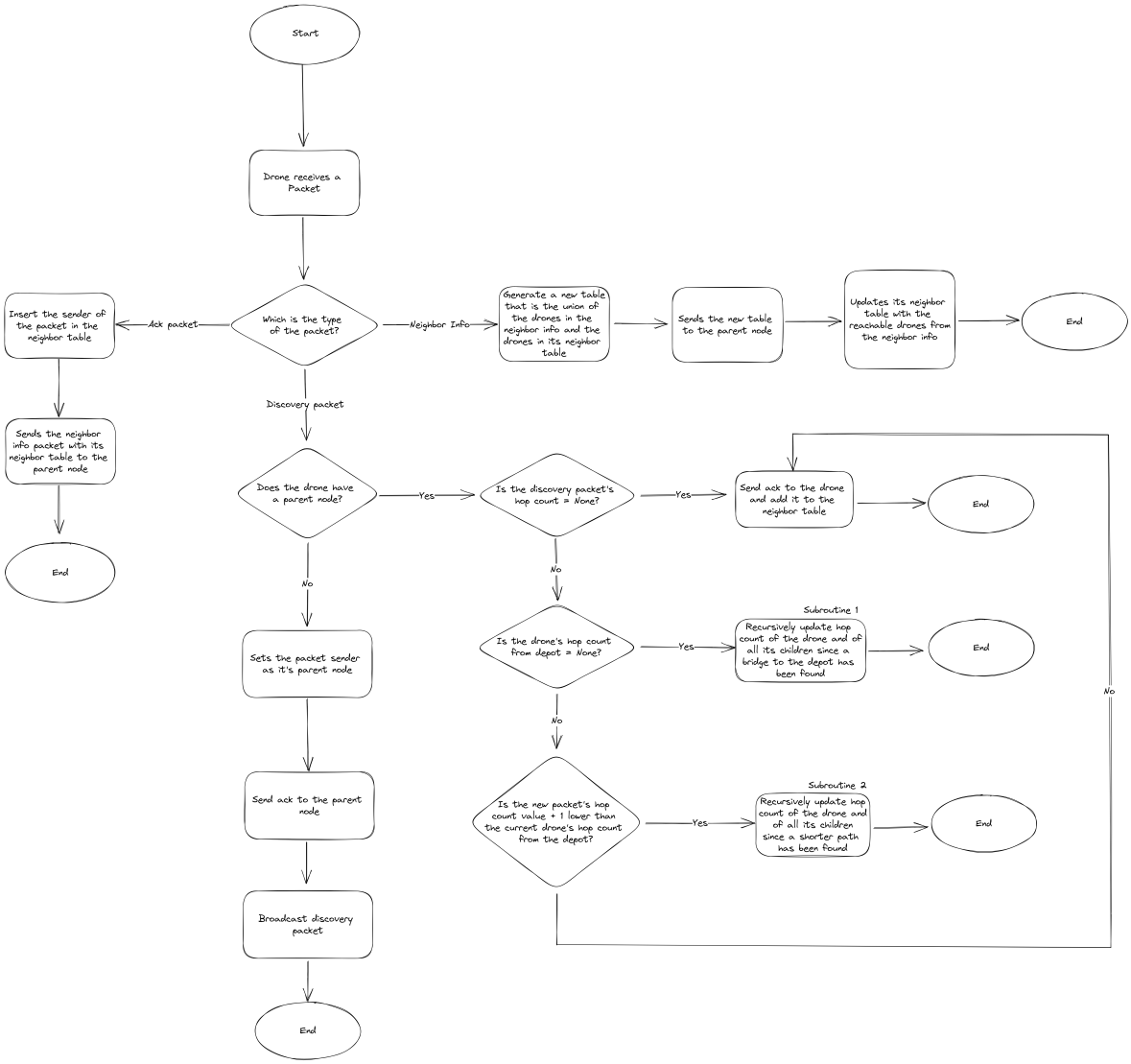

Figure 3: Drone's packet reception

2.2.1 Discovery packet reception

If the drone receives a discovery packet from another entity, which can be the depot or another drone, and it's the first discovery packet that it receives, then it sets the sender of the packet as its parent node. It then sends an ack packet to the parent node, in order to let it know that the packet has been received, and adds the parent node to its neighbor table. It then broadcasts the discovery packet to all the other drones in its communication range, after having modified the discovery packet by incrementing the hop count by 1 (if the hop count is different from None) and changing the sender with itself. In the end, it sends a copy of its neighbor table to the parent node. This is needed in order for the neighbor table to travel back recursively to the depot, only in this way, it can discover the nodes that are reachable.

If the receiver drone already has a parent node, meaning it's not the first discovery packet that he's received, or it is disconnected from the depot, then we have to distinguish the following cases:

1. The drone's stored hop count from the depot is None and the *hop* field in the discovery packet is None:
   This means that a drone not connected to the depot is sending the discovery packet to another not connected drone. The receiver sends an ack to the sender, adds it to its neighbor table, and broadcasts the discovery packet as seen before, but without updating the hop count.

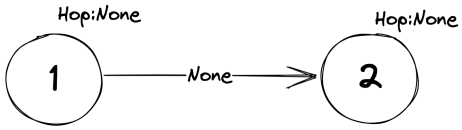

Figure 4: Case 1 (None → None)

2. The drone's stored hop count from the depot is None, but the *hop* field in the discovery packet is different from None:
   This means that the receiver is disconnected from the depot, but it's receiving a packet from a connected drone. It recursively updates its hop count information and the one of its children (see subsubsection 2.3.2), sets the sender as its parent node, and sends it the ack, before broadcasting the discovery packet with the updated hop and sender field.

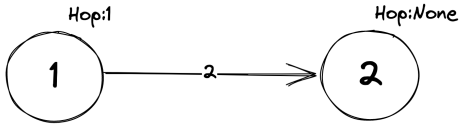

Figure 5: Case 2 (Numerical hop → None)

3. The drone's stored hop count from the depot is a higher number than the *hop* field in the discovery packet:

This means that the sender of the packet would create a shorter path between the receiver and the depot. The receiver recursively updates its hop count and the one of its children (see subsubsection 2.3.1), sets the sender as its new parent node, and sends it the ack, before broadcasting the discovery packet with the updated hop and sender field.

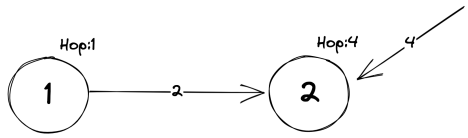

Figure 6: Case 3 (Lower hop → Higher hop)

4. The drone's stored hop count from the depot is a lower number than the number in the discovery packet's *hop* field:
   This means that the new node will be a longer path to the depot, and so it's not useful to consider it. The receiver sends an ack to let the node know that it's in the neighborhood and adds it to the neighbor table.

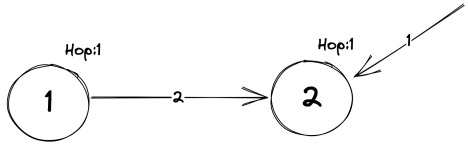

Figure 7: Case 4 (Higher hop → Lower hop)

5. The drone's stored hop count from the depot is different from None, but the *hop* field in the discovery packet is None:
   This means that a connected drone it's receiving a packet from a non-connected drone. It just sends the ack to the sender in order to let it know that it's in the neighborhood, and adds it to its neighbor table.

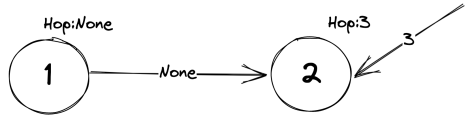

Figure 8: Case 5 (None → Numerical hop)

We can see that no matter the information inside the discovery packet and the hop count that the drone stores, every time a drone receives a discovery packet, it sends back the ack to the sender of the packet and adds it to its neighbor table. This is done in order for the drones to know which are the drones in the neighborhood.

2.2.2 Ack packet reception

When a drone receives an ack packet, it means that the sender of the packet is in its neighborhood, so it adds it inside its neighbor table and sends a copy of the neighbor table inside as a neighbor info packet to the parent node.

2.2.3 Neighbor info packet reception

When a drone receives a neighbor info packet that contains a certain neighbor table, it generates a new table that is the union between its neighbor table and the one inside the packet. It then sends the new neighbor table to the parent node, before adding the reachable nodes also inside its neighbor list.

When the depot receives a neighbor info packet then it updates its node table with the new drones that it has received. If a piece of certain drone information is duplicated, then it stores the one with the lowest hop count, because it's the most up-to-date information, since a hop update must've happened.

2.3 Hop update and bridging

2.3.1 Lower hop update

As shown in Figure 9, it can happen that a drone that already has received a discovery packet, and so that already has a parent node, receives another discovery packet from a drone that would shorten the path to the depot. In case this happens, its parent node and information of the hop count from the depot has to be changed, as well as the hop count of the nodes in the tree that has the receiver as root.

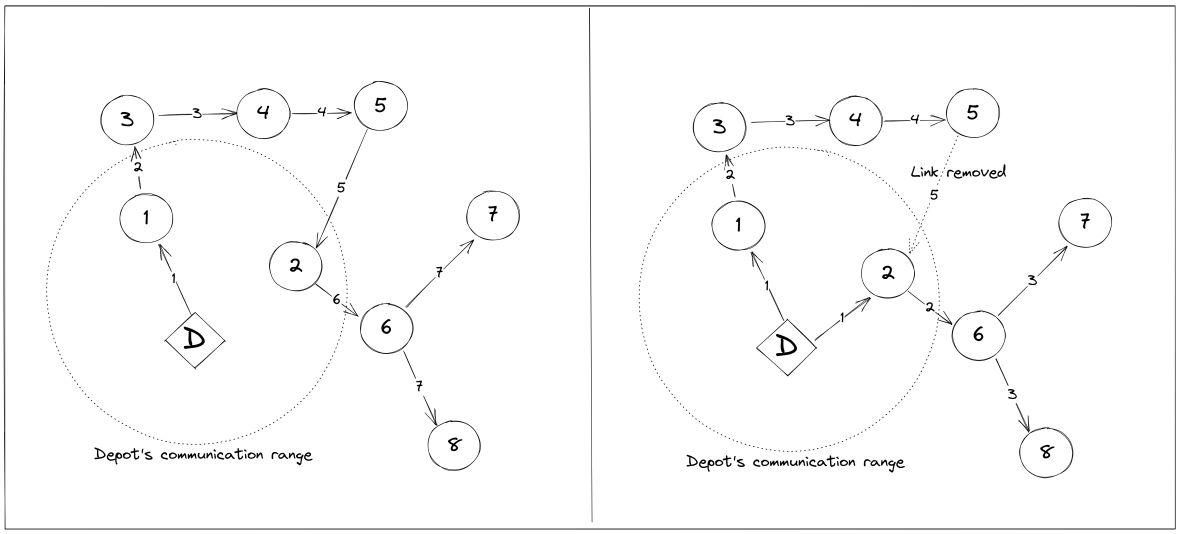

Figure 9: Lower hop count update

2.3.2 Bridging

As shown in Figure 10, it can also happen that a drone disconnected from the depot (with hop count information set to None) that already has children (exist one or more drones that have set it as parent node) receives a discovery packet from a drone that's connected to the depot, meaning the discovery packet will have the *hop* field different from None. In this case, the discovery packet sender can be used as a bridge to connect the rest of the unconnected group of drones to the depot, so the receiver's hop count and parent node must be updated, as well as the hop count of the nodes in the tree that has the receiver as root.

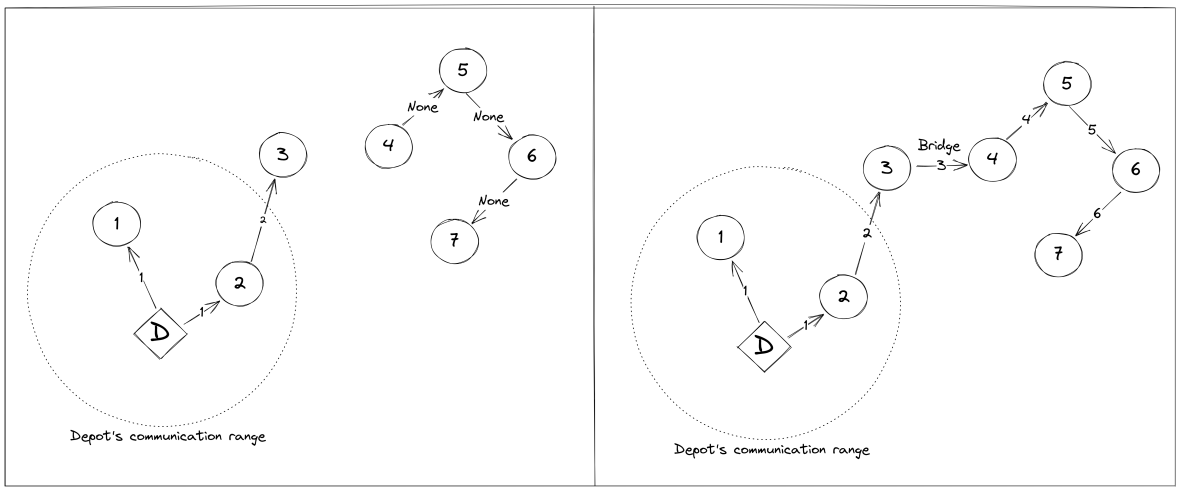

Figure 10: Bridging

2.4 Avoiding loops in the network

Since the groups of drones disconnected from the network don't have the information about the hop, they cannot "naturally" avoid a loop of father-child relationship. In order to avoid it and not cause problems, whenever a disconnected node receives a DiscoveryPacket from another node, it first checks if the sender it's in its chain of reachable nodes, and if so, it just sends an ack (to let it know that the packet arrived and that it's in its neighborhood) but doesn't add it as it's as the parent node, even if the receiver doesn't have a parent node, otherwise it will close the loop.

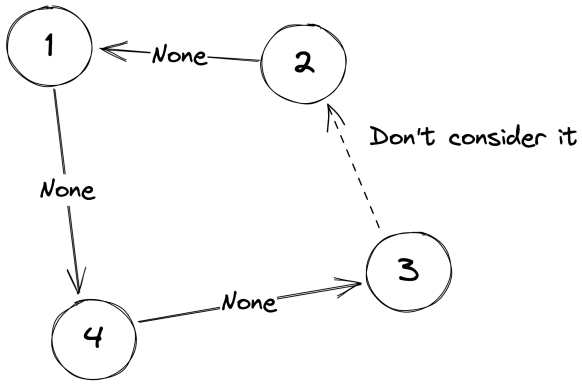

Figure 11: Avoiding loops in the network

3 Learning model

The model proposed by the paper uses a Q-Learning approach. **Q-Learning**, one of the **reinforcement learning** methods, requires low cost to learn about a given environment because it simply observes the Q-Value to make a decision. It allows the **agent** to learn the value of an action in a particular state. Q-learning finds an **optimal policy** π that maximizes the total reward.
The **reward function R(a, s)** is a function that assigns a value for every action a given the agent is in the state s. The higher the value, the better the action, and vice-versa. The **Q-table** is a data structure that contains the current value for every combination of action a and state s. Given a certain action a and a state s, the formula used to update the Q-table is the following:

$$NewQ[s_t][a_t] = (1-\alpha)\cdot Q[s_t][a_t] + \alpha \cdot \left(r_{[s_t][a_t]} + \gamma \cdot \max\left(Q[s_{t+1}]\right)\right) \tag{1}$$

where s_t means the current state and a_t means the action selected at time t, α is the learning rate that controls how much we update the Q-Value and γ is denoted as the discount factor and represents the importance of future rewards. The repetition of this equation represents the sum of rewards that will be gained in the future. In this work we assume that both states and actions are drones: a drone to select a routing path is defined as 'state', and selecting a routing path is defined as 'action'.

3.1 Relay selection

During the **relay selection** process an action is selected based on the Q-Table values. Assuming that a certain drone has some packets to send, it will send them to the neighbor (a drone in its communication range) that has the highest Q-Value. If there are no neighbors, the drone will keep the packets. Moreover, if the drone is in the communication range of the depot then packets are immediately delivered to the depot.

3.2 Reward function

Once an action is taken, a reward is computed immediately and it is used to update the Q-Table. The reward function is the following:

$$r = \begin{cases} r_{max} & \text{when } s_{t+1} \text{ is destination} \\ r_{min} & \text{when } s_t \text{ is local minimum} \\ \omega e^{\frac{1}{hop}} + (1-\omega)\text{link stability} & \text{otherwise} \end{cases} \tag{2}$$

where r represents the obtained reward, r_{max} is the maximum reward, and r_{min} denotes the minimum reward. ω is the weighting factor that balances the bias between the link stability and the number of hops and varies between 0 to 1. If there is a destination among the following states, meaning that the drone is in the communication range of the depot, then r_{max} is received to complete the data transmission. If the current state is a local minimum, meaning that the drone chosen as the best action moved even further away from the depot, then we give r_{min}. Otherwise, the reward is given in consideration of link stability and minimum hops to the destination, so that the stable state has a high Q-value while having as few hops as possible. It might happen that a path between the current drone and the depot is not defined. This is due to the fact that during the discovery process there wasn't a chain of drones between the current one and the depot that allowed the current drone to be discovered. In this case the hop is not defined (see subsubsection 2.2.1) so we only use the link stability.
Link stability is defined as follows:

$$\text{link stability}_{i,j}(t) = (1-\beta)e^{\frac{1}{v_{i,j}}} + \beta\frac{\sum_{k=t-n}^{t-1}\text{link quality}_{i,j}(k)}{n} \tag{3}$$

where, β is a coefficient weight value and n is the number of neighboring nodes.
In addition, $v_{i,j}$ represents the **relative speed** between drone i and j. The relative speed between two drones is computed as follows:

$$\text{v}_{i,j}(t) = \left|\frac{olddistance - curdistance}{stepduration}\right| \tag{4}$$

We take the absolute difference of the distance between the drones at the previous step and the current step and then divide it by the duration of a step.

Due to some limitations of the simulator, we simplified the **link quality** to be the distance between drone i and j. To compute the value of the link quality, we use an exponential decay function that has the following behavior:

$$\text{link quality}_{i,j}(t) = e^{\frac{-7\cdot\text{D(i,j)}}{\text{MCR}}} \tag{5}$$

where D(i,j) is the distance between drone i and j and MCR is the maximum communication range between two drone. The higher the distance between the drones, the lower the link quality between them.

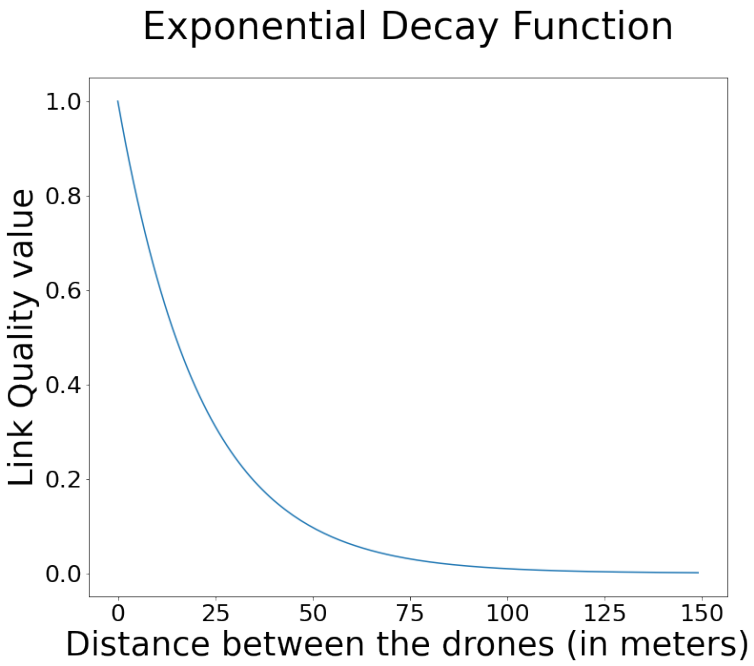

Figure 12: Behavior of the Exponential Decay Function used to calculate the Link Quality

Therefore, the link stability has a higher value when the speed at which drones i and j move away from each other is slow and the value of link quality is high.

3.3 Hyperparameter tuning

A big part of the experimentation was to change the hyperparameters learning rate α, exploration factor ϵ, and the coefficient weights β and ω in order to achieve the best results possible. After running various tests with multiple combinations of these values, we have seen that the combination that gives the best results, given our system and reward function, is:

$$\alpha = 0.8 \; \gamma = 0.1 \; \beta = 0.8 \; \omega = 0.7 \tag{6}$$

The discount factor γ represents the importance of future reward and it is set to a low value since we're in the contest of a non-stationary network, and so we don't want to give large weights to the future values.
On the other hand, the learning rate is higher, since in non-stationary networks it's better for the node to learn fast because of their high mobility.
β is used in the link stability function to weight speed and link quality. The higher the value β, the greater importance is given to link quality (at the expense of relative speed). We chose a large β because we noticed we got a greater performance when giving a larger weight to the link quality that is based on the distance between two drones.

Similarly, ω is used in the reward function to weight the number of hops and the link stability. The higher the value ω, the greater importance is given to the number of hops (at the expense of the link stability). We chose a large ω because we noticed that the number of hop, when defined, is more important to define the distance between the current drone and the depot, with respect to the link stability that simply gives information on the distance between the current drone and the previous one.

4 Simulator changes

In order to implement all the features described above, we made some **changes to the simulator**.
The **first modification** concerned **the depot**: The *Q-learning based routing protocol* proposed by the paper assumes the depot to be an **active actor** meaning that it is the one that initializes the nodes discovery at each step. In the original implementation of the algorithm, the depot was a passive actor and its aim was to merely receive the packets sent by the drones in its communication range so we changed the behavior of the depot in the *uav_entities* file. We assume all the depot to be capable of sending control packets over a long distance. So, at each simulation step, the depot sends each drone in the network a discovery message to start the stepwise node discovery. Every node that receives the message starts the discovery accordingly (see section 2). From now on, the depot only stores the data received by the drones with which there exists a route, so it becomes again a passive actor until the end of the current simulation step.
The **second modification** affected **drones**: the routing protocol requires each drone to compute the link stability between itself and its neighbors at each simulation step. We change the drone behavior in the *uav_entities* file in a similar way to the depot: at each step every drone computes the link stability with the neighbors found during the discovery process. All these values are stored in memory (each drone has its local information) and used to update the Q-Value in the reward function.
We also changed the way the reward function is triggered. Previously, a feedback was given every time a packet was delivered to the depot (positive feedback) or a packet got lost (negative feedback), and the reward was computed accordingly. On the contrary, the routing protocol in the paper requires to compute the reward when an action is taken in a specific state, so at each step. For this reason we call the reward function at each simulation step.

5 Experiments

After implementing the *Q-learning stepwise routing protocol*, we made a few experiments varying the number of drones and testing with different seeds to see how it performed. We evaluated the algorithm by considering three metrics: **mean number of relays**, **packet mean delivery time** and **packet mean delivery ratio**. The final result is the following:

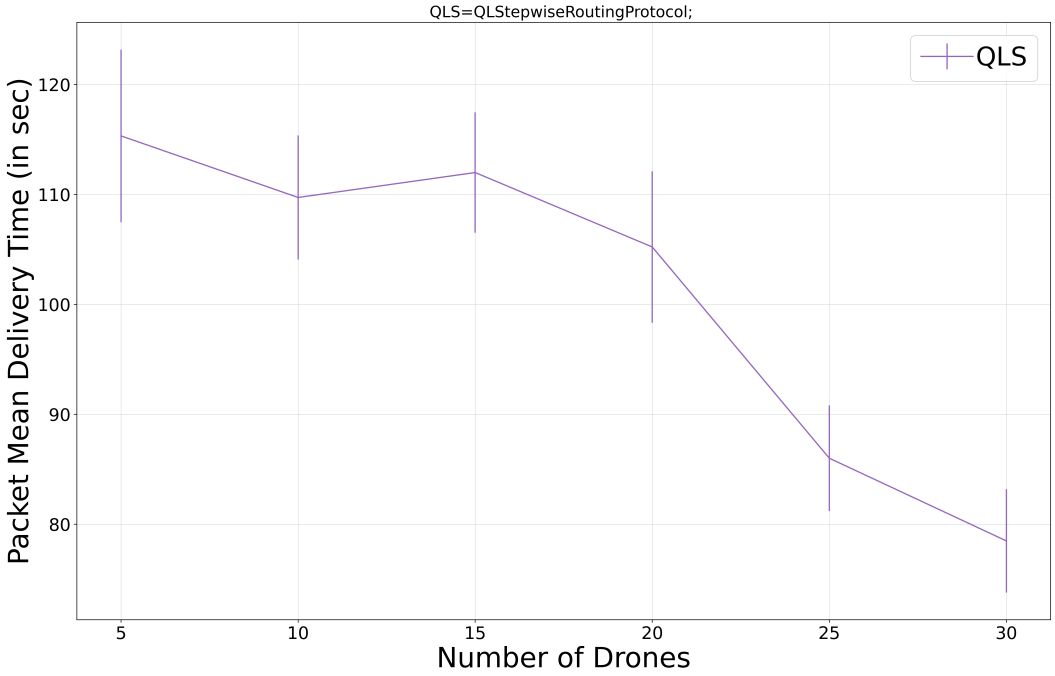

Figure 13: Packet mean delivery time in Q-learning stepwise routing protocol

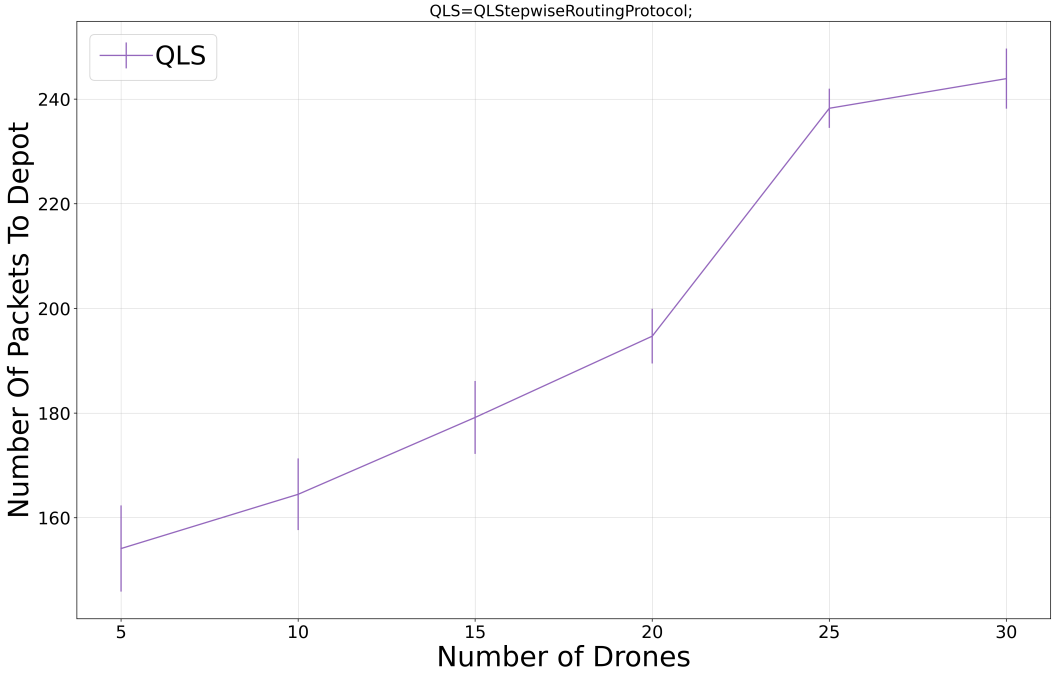

Figure 14: Packet mean delivery ratio in Q-learning stepwise routing protocol

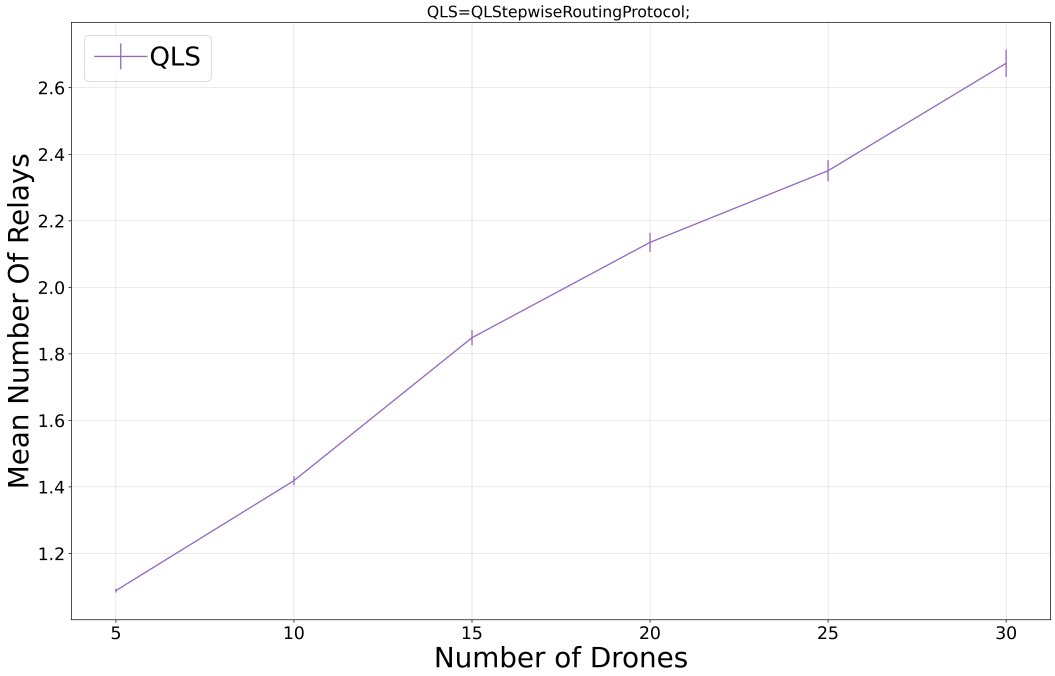

Figure 15: Mean number of relays in Q-learning stepwise routing protocol

It is immediate to notice that our implementation of the *QLS protocol* has a low packet delivery ratio with a low number of drones. As we already experienced in hw1, only using values from the Q-table to select the best action is not enough when using few drones. In general, we noticed that the Q-table is updated infrequently due to the fact that drones rarely cross each other. It may happen that when selecting an action, we don't have enough information to decide which is the best one and eventually choose the worst one (e.g. a drone that is moving away even far away from the depot or has a worse link stability). The situation described so far should get better in time as drones "increase their experience" but, since the Q-table is rarely updated, it doesn't seem to happen. Contrarily, with a higher number of drones the *packet delivery ratio* is much higher thanks to the fact the Q-table values are properly updated and the very best action is taken most of the time.
As expected, the packet mean delivery time decreases and the mean number of relays increases with a higher number of drones. Packets are exchanged properly among drones and they are delivered to the depot faster. Let's now compare the *Q-learning stepwise routing protocol* with the **Geographic Routing protocol**, the **Random Routing protocol** and our q-learning solutions for homework 1 (**Simple QLRouting protocol** and **GEOQL Routing Protocol**):

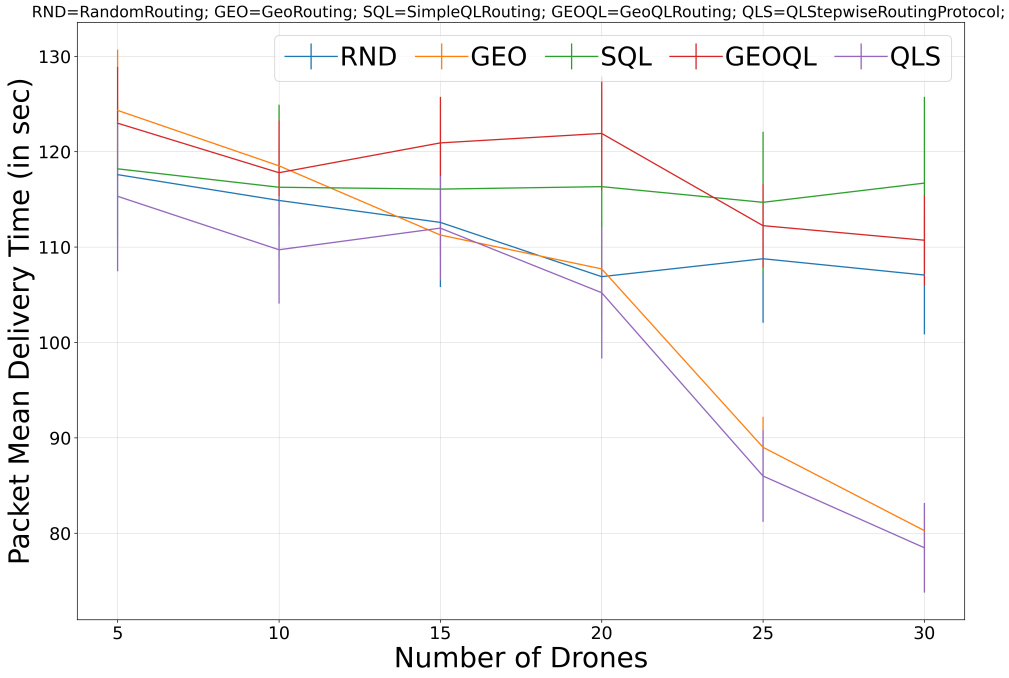

Figure 16: Packet mean delivery time comparison with all algorithms

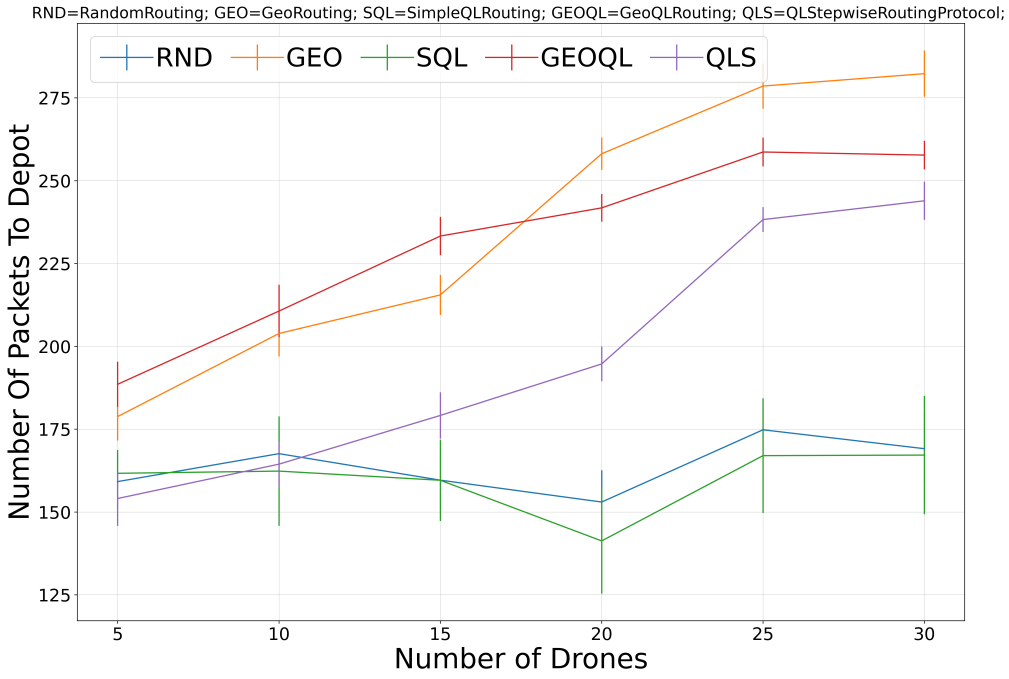

Figure 17: Packet mean delivery ratio comparison with all algorithms

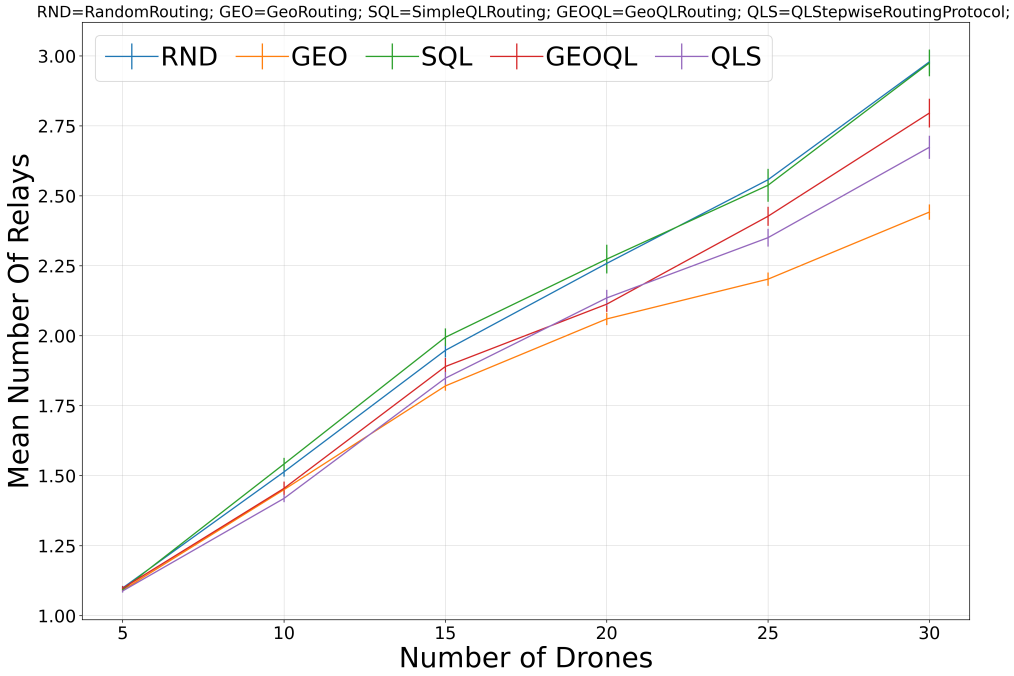

Figure 18: Mean number of relays comparison with all algorithms

As already seen before, the QLS algorithm is pretty inefficient with a low number of drones due to the fact that the Q-Table is not properly updated. The same problem also happens in the SQL algorithm (hw1) that uses only Q-Values to select the best action. This results in a packet delivery ratio that is more or less the same as the random approach. When introducing geographic information, the situation gets slightly better in fact the GEOQL algorithm (hw1) filter uses the Q-Values to select the best action among the drones which are moving to the depot. With a higher number of drones, the QLS algorithm performs way better but no more than the geographic approaches.
For what concerns the *packet mean delivery time*, the QLS algorithm is the best approach with almost every number of drones, meaning that it is able to deliver packets pretty fast. When comparing the mean number of relays, we can see how they perform mostly the same except when considering a higher number of drones: in this case the QLS algorithm can be leveled to the geographic approaches.

5.1 GeoQLS

Given that the geographic approach has the best performance regarding the packet delivery ratio, we tried to create an hybrid algorithm between the GEOQL and the QLS one. The new algorithm works exactly like our implementation of the *QLearning Stepwise Routing protocol* but, when selecting the best action, it chooses the drone with the highest Q-Table value among the ones which are heading towards the depot. Implementing this modification, the performance highly increases:

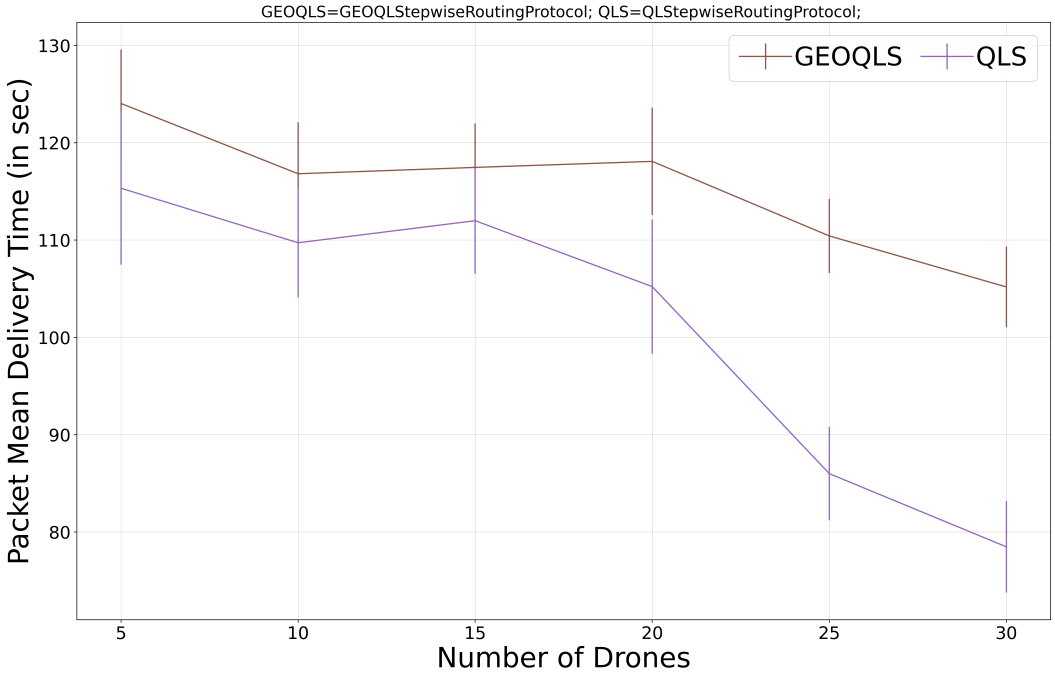

Figure 19: Packet mean delivery time comparison between QLS and GEOQLS

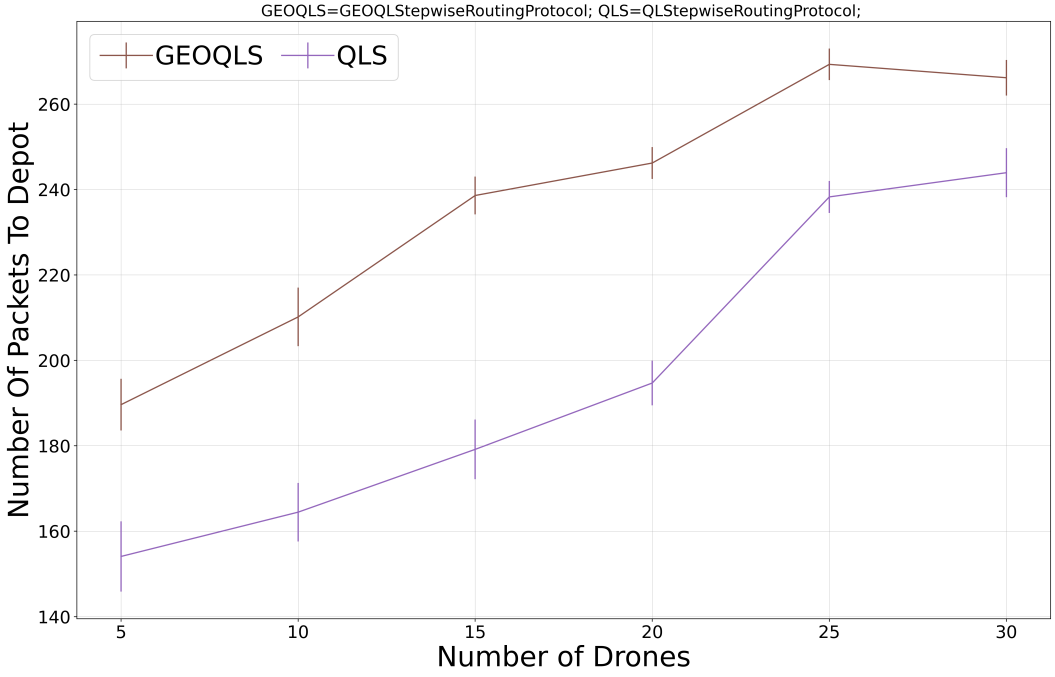

Figure 20: Packet mean delivery ratio comparison between QLS and GEOQLS

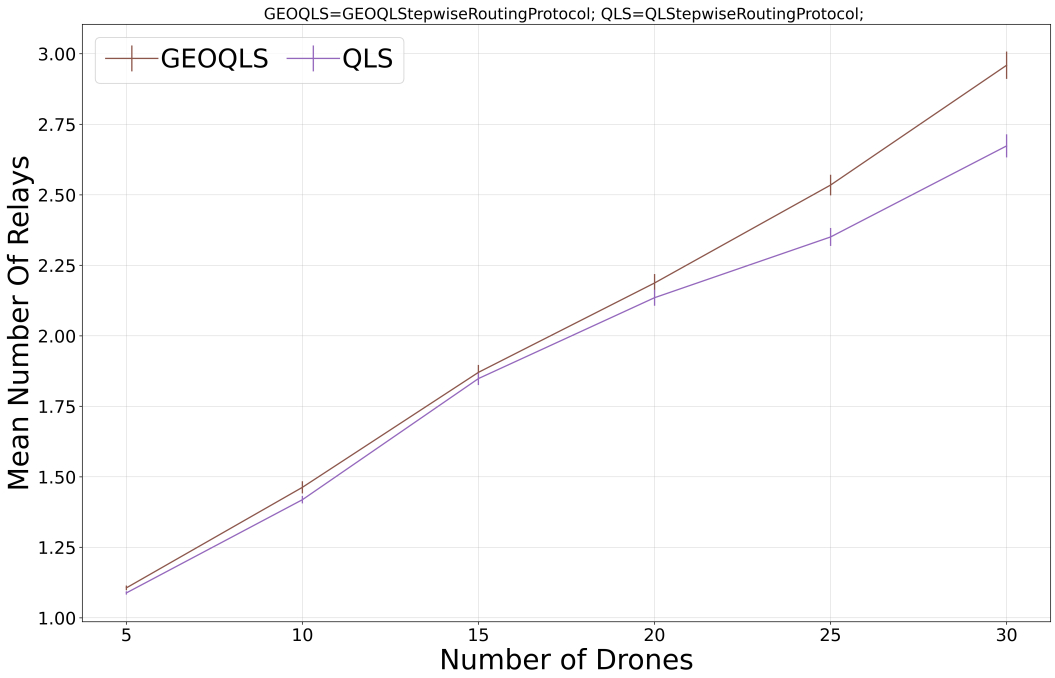

Figure 21: Mean number of relays comparison between QLS and GEOQLS

Thanks to the introduction of some geographic information during the relay selection process, the packet delivery ratio highly increases, especially with a low number of drones. Unfortunately, the packet mean delivery time increases consequently. Let's compare the GEOQLS with the other geographical approaches:

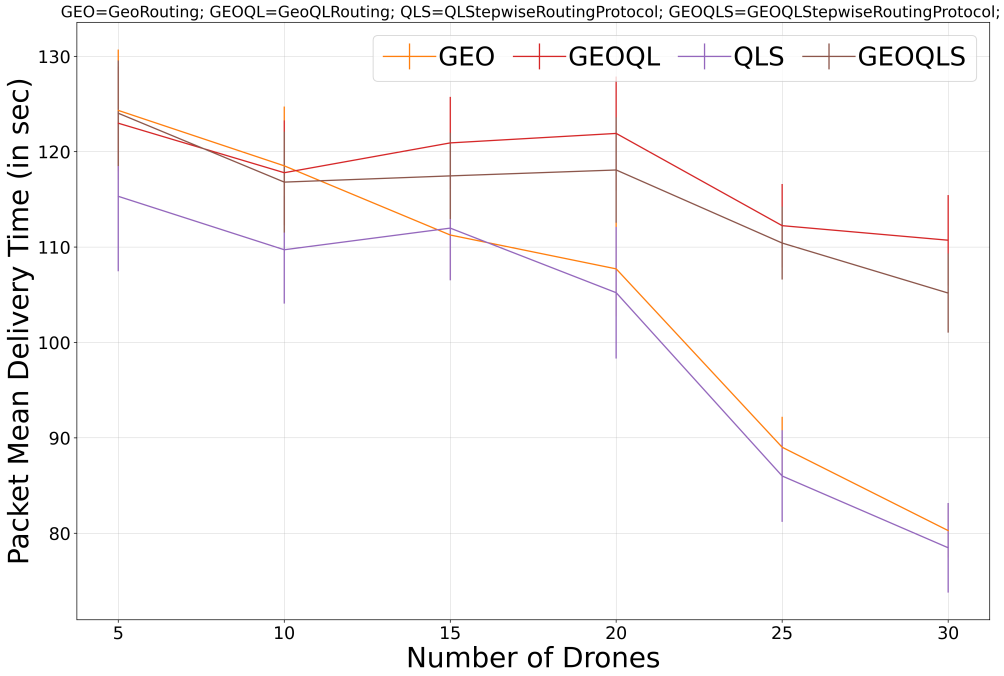

Figure 22: Packet mean delivery time comparison between GEOQLS, QLS, GEO and GEOQL (hw1)

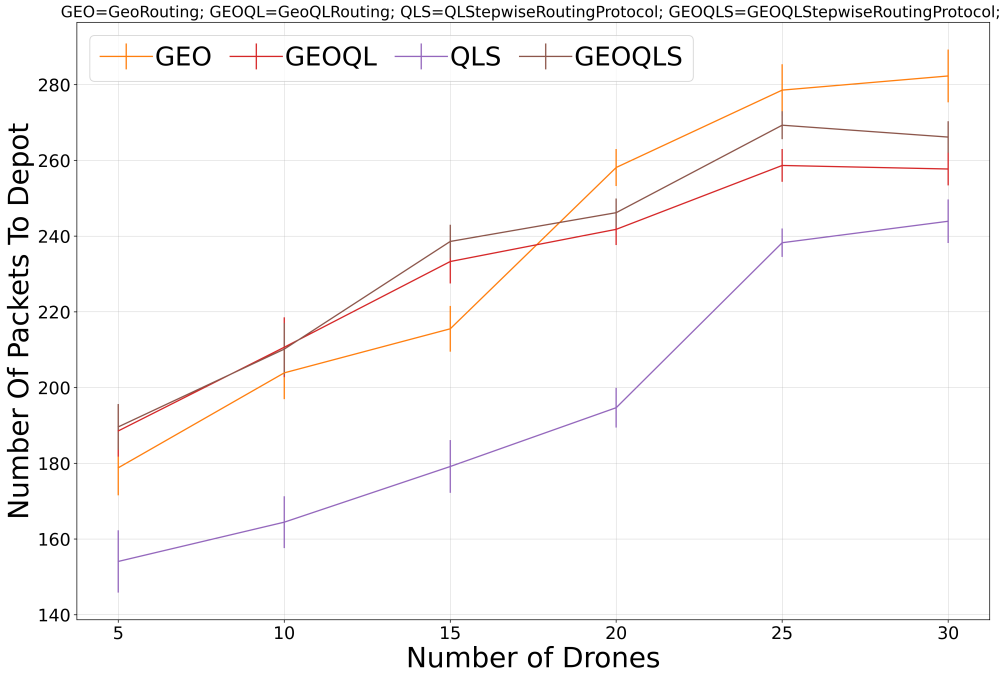

Figure 23: Packet mean delivery ratio comparison between GEOQLS, QLS, GEO and GEOQL (hw1)

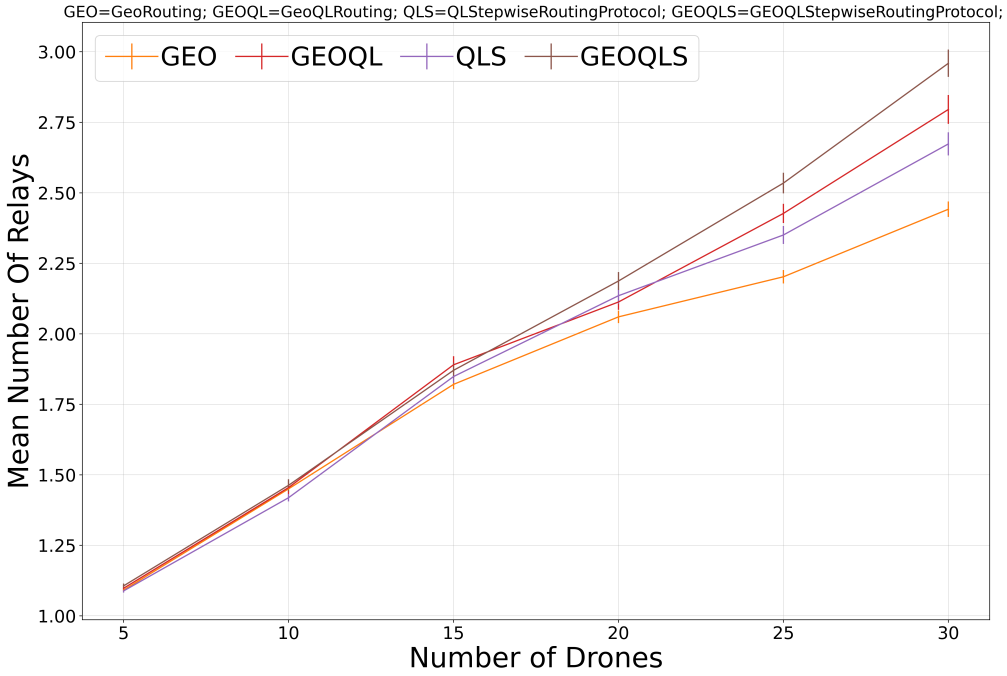

Figure 24: Mean number of relays comparison between GEOQLS, QLS, GEO and GEOQL (hw1)

Concerning the number of packets to depot, GEOQLS performs better than QLS thanks to the exploitation of the geographical information, especially with a low number of drones. It also performs better than GEO in most of the cases and GEOQL with a high number of drones meaning that the more complex reward function of the QLS algorithm helps when deciding the action to choose. As seen before, the packet delivery time is slightly higher as well as the mean number of relays.
GEOQSL is a good compromise because it works best with a low number of drones and still has a great packet delivery ratio with a higher number of drones, with a slightly higher packet delivery time.

6 Conclusions

Following all the experiments carried out, we can say that the *QLearning Stepwise Protocol* is more suitable when using a high number of drones in the network, otherwise it is preferable to use a much computationally lighter approach. We implemented the protocol following the requirements in the paper but we end up making some simplifications, such as in the link quality calculation: the original approach was to use the packet transmission time and the packet delivery ratio to compute the link quality between two drones but we had to change this idea because it was challenging to implement these features in a simulator. An extension of what we have done could be to try to simulate the packet transmission time and use this information instead of the distance between two drones and see if it performs better than our results. Still, our implementation works well with a higher number of drones and has a really low packet mean delivery time so it could be a valid choice when making large on-field operations.
Moreover, adding some geographical information when selecting the best action, makes the *QLearning Stepwise Protocol* more efficient with a slightly higher packet delivery time making it a good compromise between the pure geographic approach and the one based on q-learning.

Contributions

Alessio Lucciola First implementation of the discovery phase (start of discovery, reception of packet (discovery and ack), neighbor and node tables update), Q-learning model (link quality, link stability, reward function), hyperparameter tuning, GEOQLS experiment and report.

Danilo Corsi First implementation of the discovery phase (start of discovery, reception of packet (discovery and ack), neighbor and node tables update), Q-learning model (relative speed in link stability) and report.

Domiziano Scarcelli Improvements in the discovery phase, hop update, bridging, network loops avoidance, and report.