SIMULATING CSMA/CA WITH RL

Davide Avolio

***Abstract*—The goal of this project is to create a simulation for the CSMA/CA protocol. Two versions were created: a standard one and an adaptive one with a Reinforcement Learning approach. The two methods were compared according to some metrics such as the throughput and the packet delivery ratio.**

I. The protocol

In this project, I present a simulation of the CSMA/CA protocol both in the classical and in the adaptive version with Reinforcement Learning.

In the Carrier-sense multiple access with collision avoidance (CSMA/CA) protocol, nodes attempt to avoid a collision by transmitting only when the channel is sensed to be free. However, due to several issues such as the hidden terminal problem, the protocol was extended with the usage of the Request To Send (RTS) and Clear To Send (CTS) packets: when a node wants to send a packet and senses the channel free, it first sends an RTS to the receiver and starts to transmit only after having received a CTS. After having received the packet, the receiver sends an ACK back. In this way the hidden terminal problem is solved, indeed, when a node receives an RTS, it reads the length of the data and delays all the transmission for that amount of time (the timer is set in the NAV).

Nonetheless, collisions can still occur, especially if multiple nodes send RTS packets at the same time. The difference between the two protocols is in this step: in the classic one, all the nodes wait for a random amount of time from 0 to a contention window and then send their packet (the first one who finishes will be the one to send) and in case of collision, the contention window is increased; in the adaptive one, each node has a reinforcement learning agent that learns how to deal with collisions and makes the node wait a proper amount of time.

In this simulation, the RL algorithm that I chose is the Q-learning one. In Q-learning, every time we compute a new action, we store the reward and then we evaluate the TD error, the difference between the discounted best next Q-state and the current one.

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

II. Simulation

The simulation of the two protocols was developed in Python. It has some tunable parameters, namely:

- Number of nodes
- Packet length
- Packet generation probability
- Number of timesteps of the simulation

A. Classic CSMA/CA

The first step was to create a simulation of the classic CSMA/CA protocol[1].

I created a class for the transmitter node, a class for the receiver node and a class for each kind of packet (data, RTS, CTS and ACK).

The first thing the transmitter node does is to generate a packet according to the probability specified above. Then it checks the NAV, the backoff and finally the channel. There are in total 12 checks it performs at each timestep.

The receiver node is always listening to the channel and it just sends the CTS, ACK or nothing in case there are no RTS or in case there's a collision between RTS.

The RTS packet contains the sender id and the amount of time the other nodes should wait in NAV. The CTS packet has just the destination id. The data packet has the sender id and a flag that is marked only when the data is finished (to notify the receiver). The ACK packet has instead only the destination id.

B. Adaptive CSMA/CA

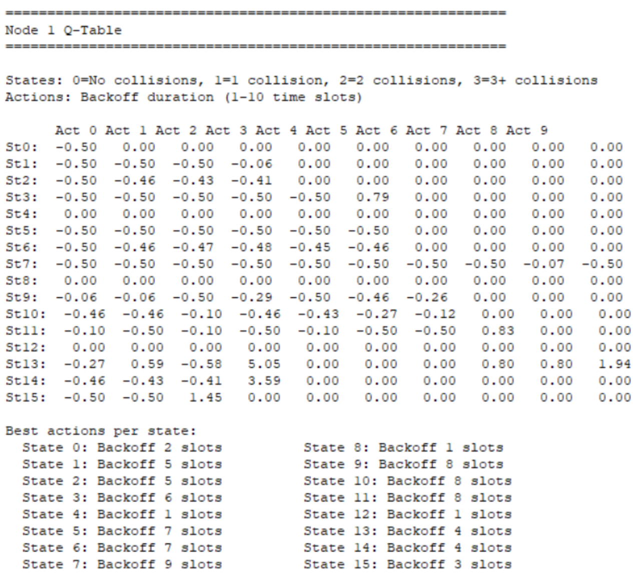
In the Adaptive CSMA/CA[2], another class was added: the RL agent. There is one agent per node that is called only in two situations: when there's a collision (negative reward) and when there's a successful transmission (positive reward). The agent replaces the exponential backoff, so the possible actions are the amount of time the node should wait. The state is instead given by the number of collisions since the last successful transmission.

Since each node is independent, they all have a separate Q-table. The states are 16 are they are based on possible ranges of recent collisions:

- 0 recent collisions.
- 1 recent collision.
- 2 recent collisions.
- 3 or more recent collisions.

And on ranges of packets in queue:

- 0 to 3 packets in queue.
- 4 to 7 packets in queue.
- 8 to 15 packets in queue.
- 16 or more packets in queue.



```
============================================================
Node 1 Q-Table
============================================================

States: 0=No collisions, 1=1 collision, 2=2 collisions, 3=3+ collisions
Actions: Backoff duration (1-10 time slots)

      Act 0 Act 1 Act 2 Act 3 Act 4 Act 5 Act 6 Act 7 Act 8 Act 9
St0:  -0.50   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
St1:  -0.50  -0.50  -0.50  -0.06   0.00   0.00   0.00   0.00   0.00   0.00
St2:  -0.50  -0.46  -0.43  -0.41   0.00   0.00   0.00   0.00   0.00   0.00
St3:  -0.50  -0.50  -0.50  -0.50  -0.50   0.79   0.00   0.00   0.00   0.00
St4:   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
St5:  -0.50  -0.50  -0.50  -0.50  -0.50  -0.50   0.00   0.00   0.00   0.00
St6:  -0.50  -0.46  -0.47  -0.48  -0.45  -0.46   0.00   0.00   0.00   0.00
St7:  -0.50  -0.50  -0.50  -0.50  -0.50  -0.50  -0.50  -0.50  -0.07  -0.50
St8:   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
St9:  -0.06  -0.06  -0.50  -0.29  -0.50  -0.46  -0.26   0.00   0.00   0.00
St10:  -0.46  -0.46  -0.10  -0.46  -0.43  -0.27  -0.12   0.00   0.00   0.00
St11:  -0.10  -0.50  -0.10  -0.50  -0.10  -0.50  -0.50   0.83   0.00   0.00
St12:   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00   0.00
St13:  -0.27   0.59  -0.58   5.05   0.00   0.00   0.00   0.80   0.80   1.94
St14:  -0.46  -0.43  -0.41   3.59   0.00   0.00   0.00   0.00   0.00   0.00
St15:  -0.50  -0.50   1.45   0.00   0.00   0.00   0.00   0.00   0.00   0.00

Best actions per state:
  State 0: Backoff 2 slots          State 8: Backoff 1 slots
  State 1: Backoff 5 slots          State 9: Backoff 8 slots
  State 2: Backoff 5 slots          State 10: Backoff 8 slots
  State 3: Backoff 6 slots          State 11: Backoff 8 slots
  State 4: Backoff 1 slots          State 12: Backoff 1 slots
  State 5: Backoff 7 slots          State 13: Backoff 4 slots
  State 6: Backoff 7 slots          State 14: Backoff 4 slots
  State 7: Backoff 9 slots          State 15: Backoff 3 slots
```

Fig. 1. QTable of a node

C. *Visualization*

All the actions of the nodes and the status of the channel are saved in separate lists, that have as many items as timesteps. To visualize these data, I used the Tkinter library. Each timestep is visualized as a cell and the color is given by the type of action performed.

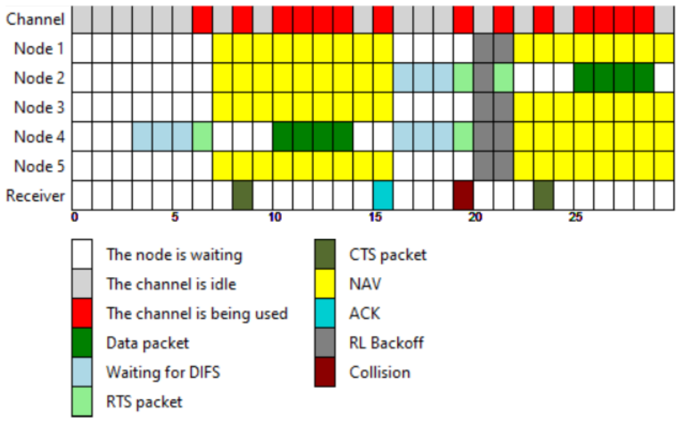


Fig. 2. An example of simulation with the classic CSMA/CA, 5 nodes and packet probability = 0.03

III. Performance

In this section we are going to analyze how the performance of a single protocol varies by changing the parameters[3]. Three metrics were considered:

- **Packets sent** The total number of packets that have been successfully sent. Since the packets have a fixed length and we're comparing simulations with the same random seed and with the same time length, this is equivalent to the throughput (only the values change, but the graph is the same).
- **Packet delivery ratio** The number of packets that have arrived successfully over the total number of packets generated.
- **Collisions** The number of collisions that occurred in the simulation.

In the following statistics, we are taking into account the average of these metrics per node.

A. *Classic CSMA/CA*

In the Classic CSMA/CA protocol there're two parameters that we can change: the number of nodes and the packet generation probability.

We start with the number of nodes:

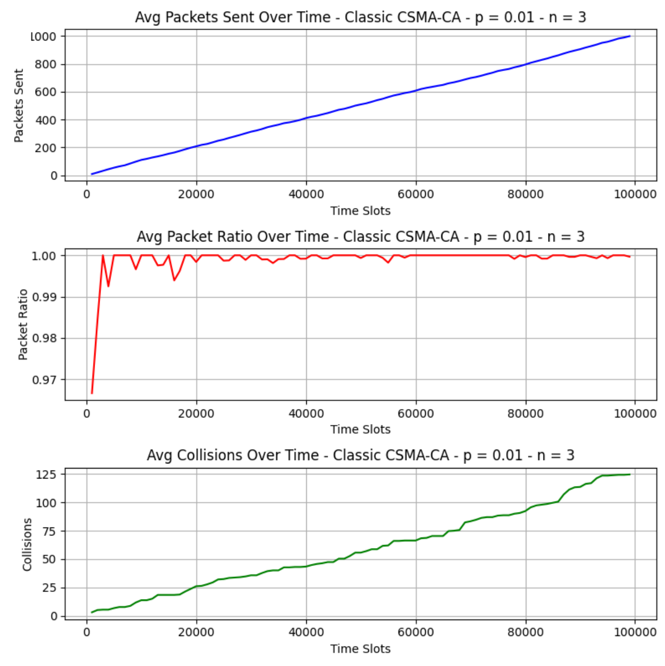


Fig. 3. Classic CSMA/CA with 3 nodes and packet probability = 0.01

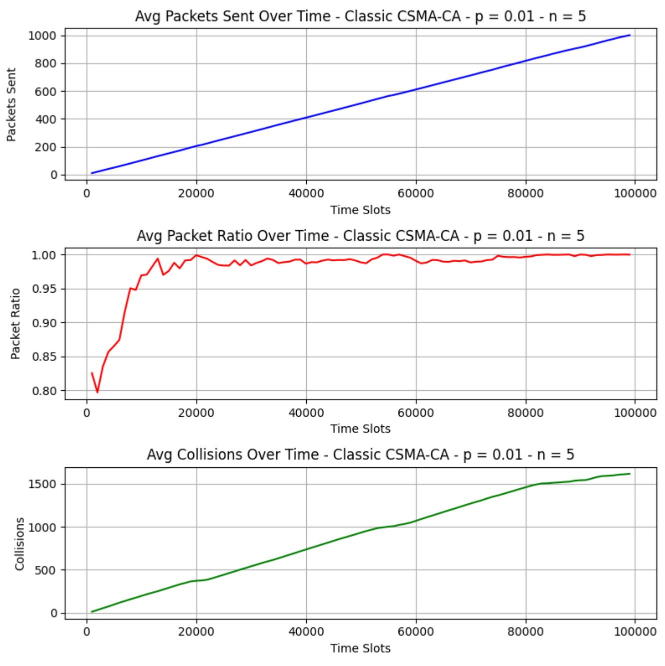


Fig. 4. Classic CSMA/CA with 5 nodes and packet probability = 0.01

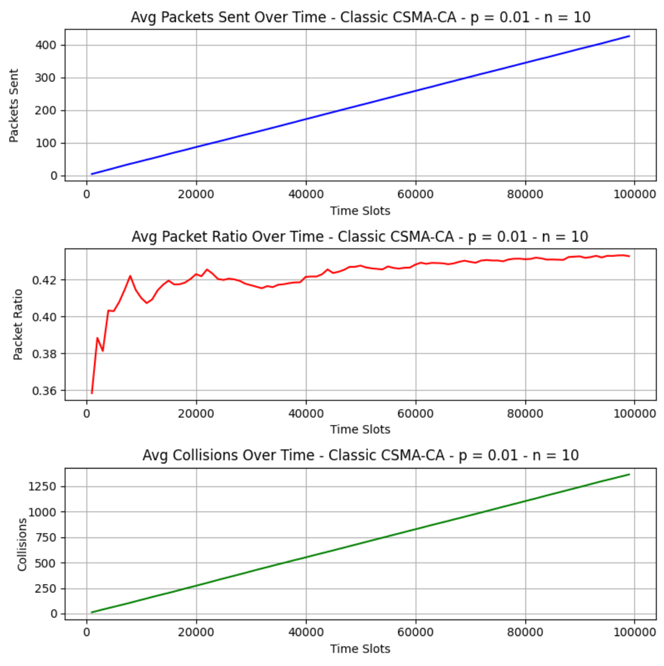


Fig. 5. Classic CSMA/CA with 10 nodes and packet probability = 0.01

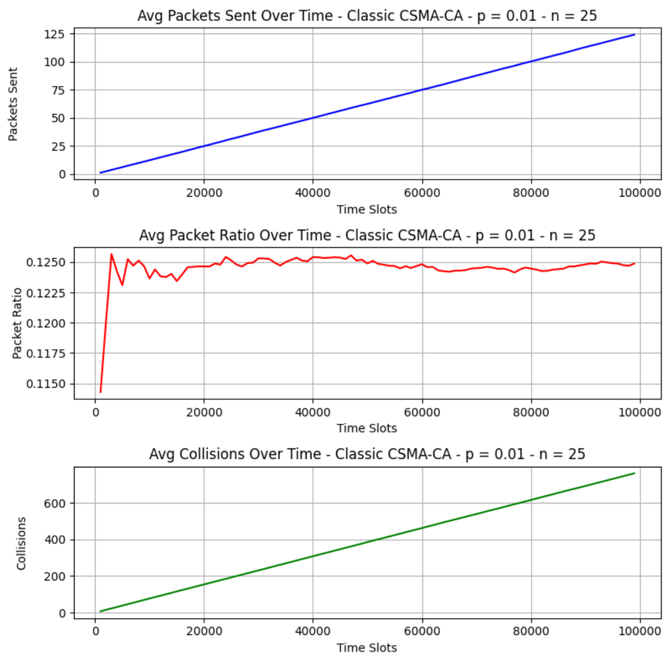


Fig. 6. Classic CSMA/CA with 25 nodes and packet probability = 0.01

We can see that the higher the number of nodes and the lower is the packet ratio and the higher are the collisions. With 3 nodes the ratio is almost always 1, which means that the nodes are managing to send all the packets in their queues. When we arrive at 25 nodes, their packet ratio is almost constant at 0.1250.

The number of packets over time also decreases, due to the fact that nodes are busy sending the same packet over and over again because of the collisions.

If we vary the packet generation probability instead:

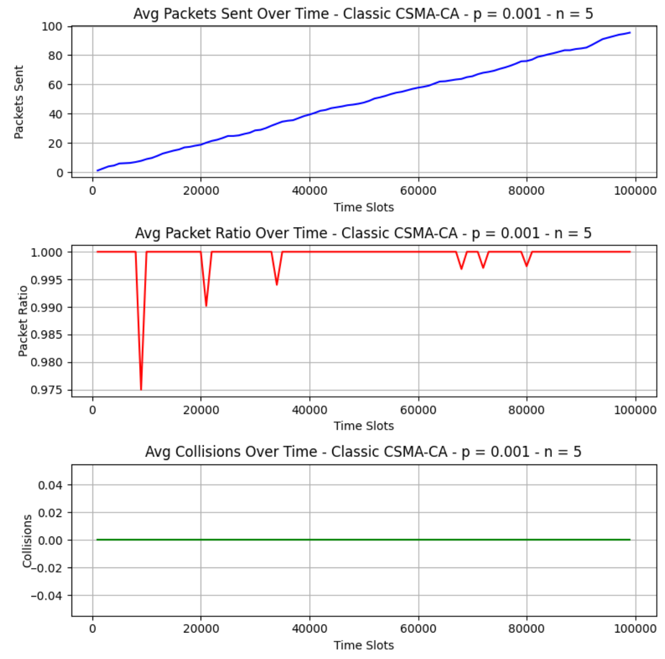


Fig. 7. Classic CSMA/CA with 5 nodes and packet probability = 0.001

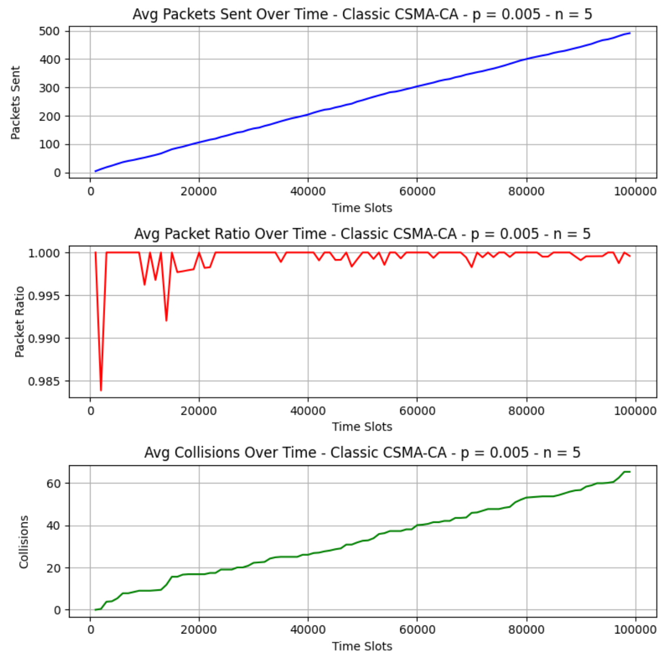


Fig. 8. Classic CSMA/CA with 5 nodes and packet probability = 0.005

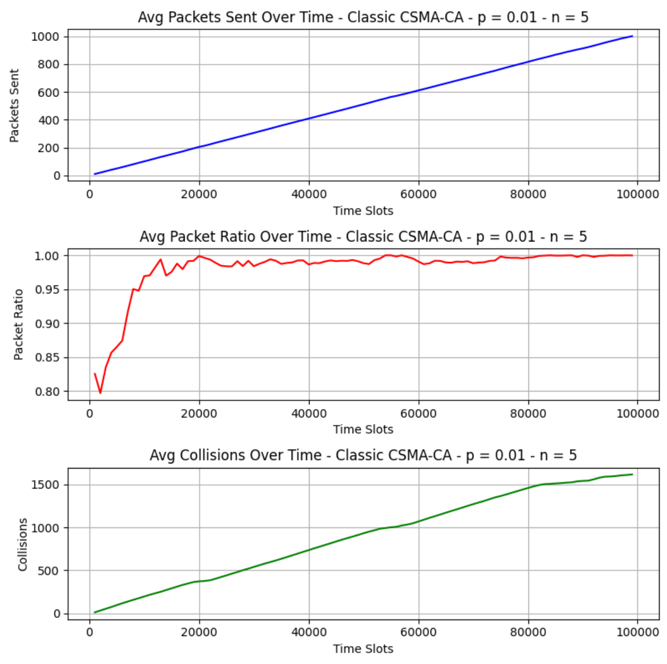


Fig. 9. Classic CSMA/CA with 5 nodes and packet probability = 0.01

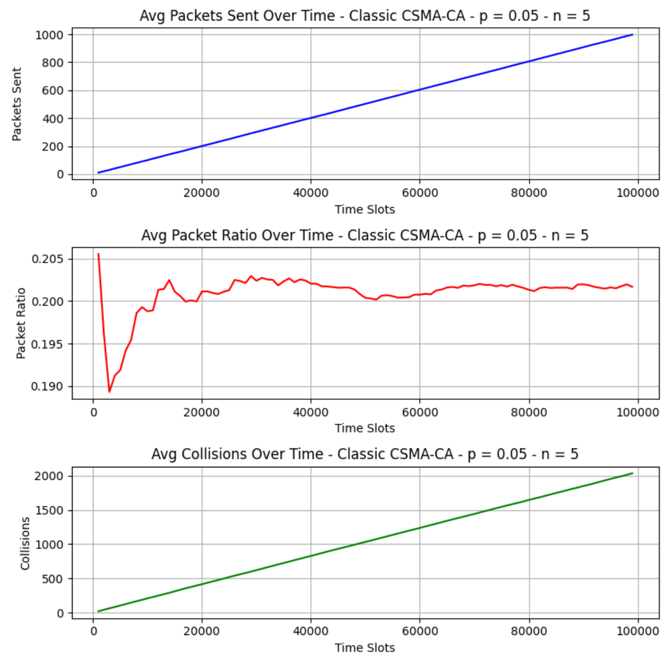


Fig. 10. Classic CSMA/CA with 5 nodes and packet probability = 0.05

We can notice that when the probability is low (0.001), the packets are generated so rarely that there are no collisions and that sometimes the channel is completely idle (the lower spikes in the packet ratio). When we increase the probability, we notice an increase in the number of packets sent, from

100 to 500 to 1000 and in collisions too. However, when the probability is too high, there are too many collisions and the number of packets sent doesn't increase anymore; we can also see the effect with the packet ratio, that is around 0.2.

B. Adaptive CSMA/CA

In the Adaptive CSMA/CA protocol there're three parameters that we can change: the number of nodes, the packet generation probability and the exploration coefficient (ϵ). We start with the number of nodes:

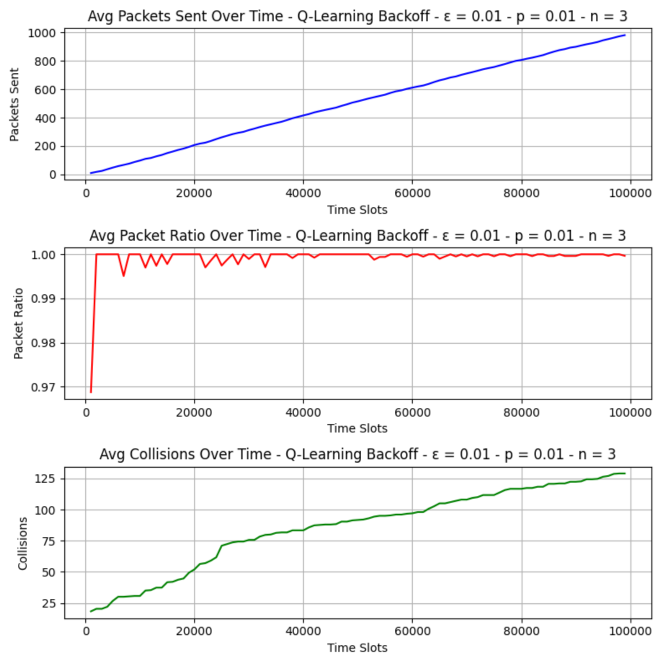


Fig. 11. Adaptive CSMA/CA with 3 nodes, packet probability = 0.01 and ϵ = 0.01

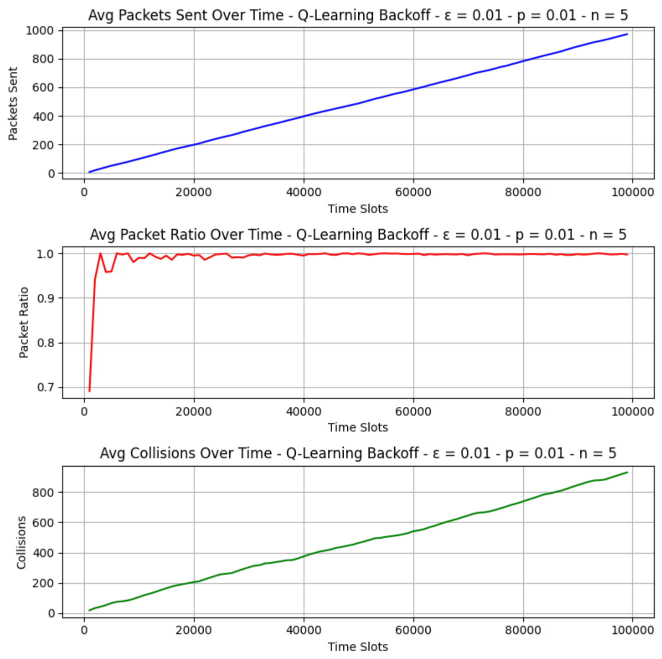


Fig. 12. Adaptive CSMA/CA with 5 nodes, packet probability = 0.01 and ϵ = 0.011

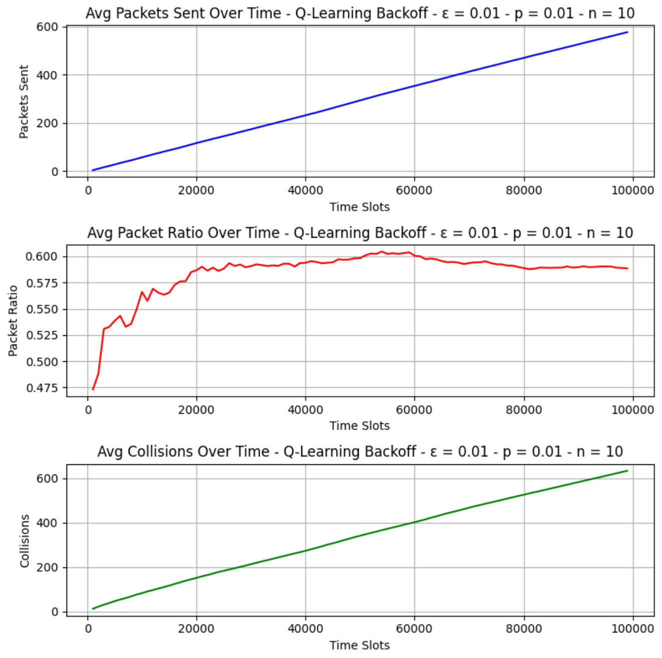


Fig. 13. Adaptive CSMA/CA with 10 nodes, packet probability = 0.01 and ϵ = 0.01

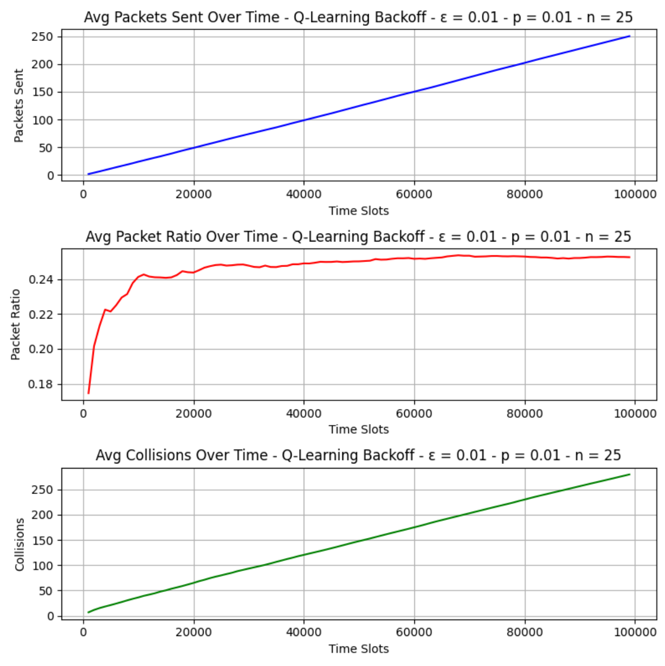


Fig. 14. Adaptive CSMA/CA with 25 nodes, packet probability = 0.01 and ϵ = 0.01

Just like in the Classic CSMA/CA, the higher the number of nodes and higher is the number of collisions and the lower is the packet ratio. But in general, the Adaptive CSMA/CA, manages to send more packets with the same conditions.

If we consider the packet probability instead:

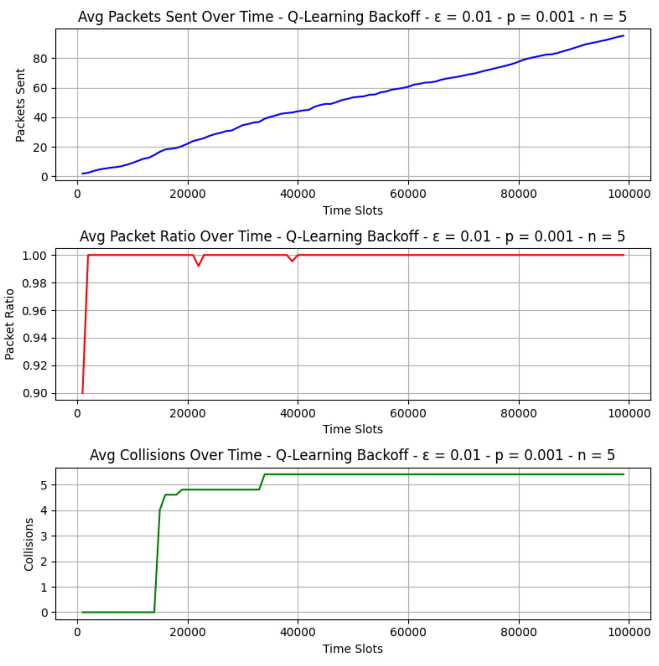


Fig. 15. Adaptive CSMA/CA with 5 nodes, packet probability = 0.001 and ϵ = 0.01

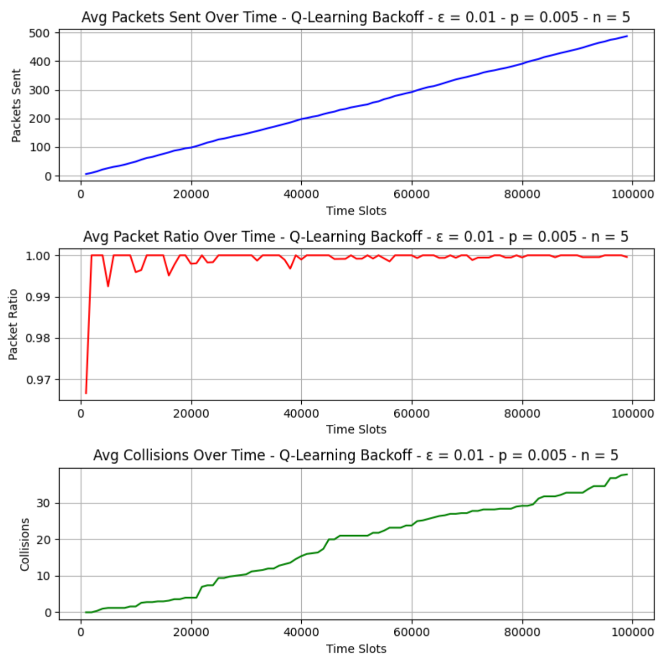


Fig. 16. Adaptive CSMA/CA with 5 nodes, packet probability = 0.005 and ϵ = 0.01

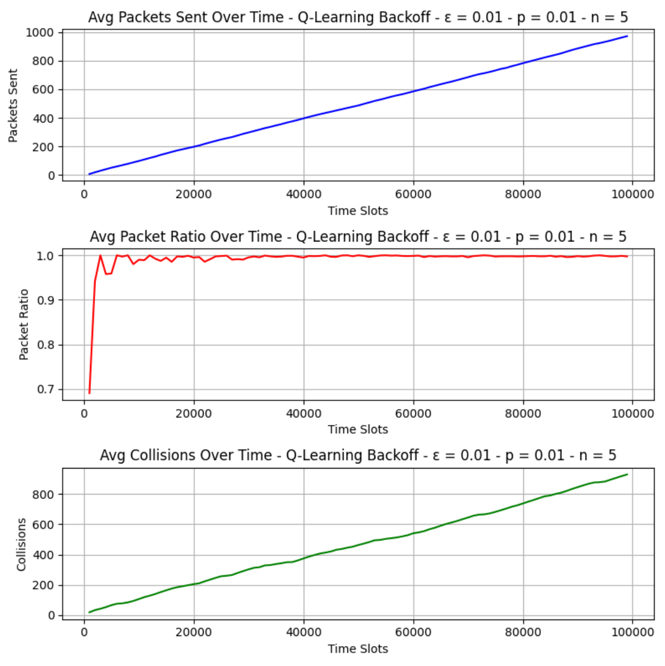


Fig. 17. Adaptive CSMA/CA with 5 nodes, packet probability = 0.01 and ϵ = 0.01

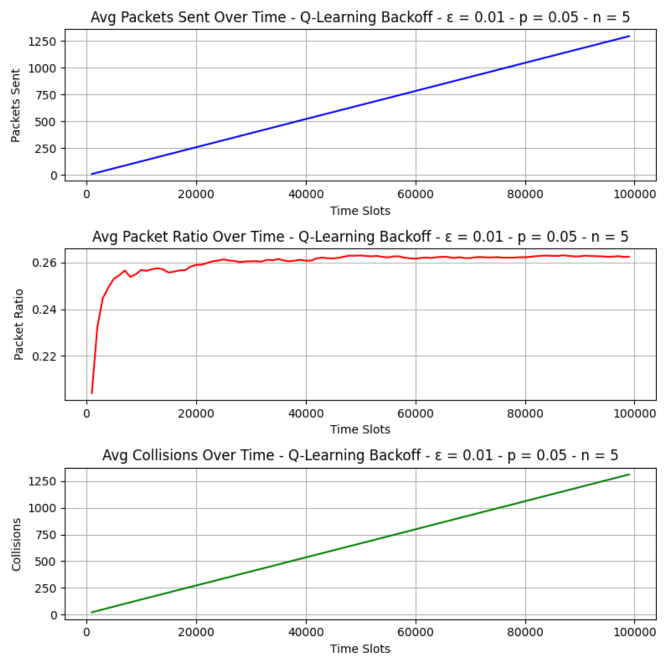


Fig. 18. Adaptive CSMA/CA with 5 nodes, packet probability = 0.05 and ϵ = 0.01

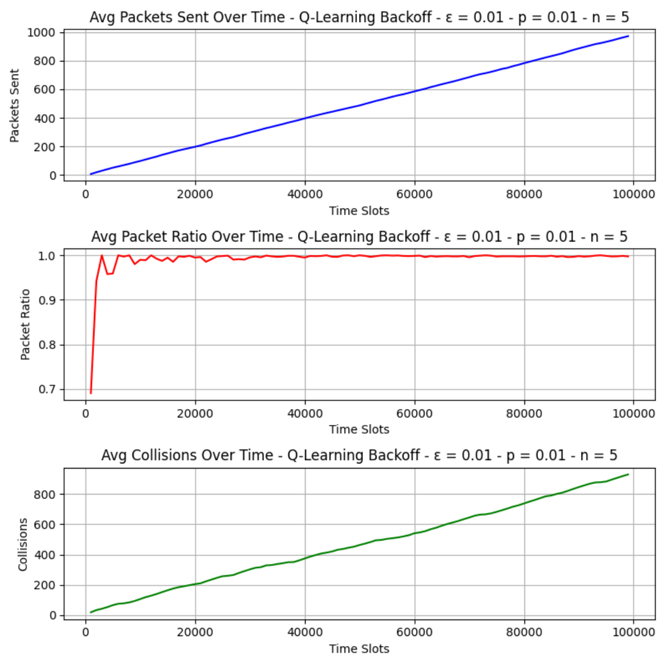


Fig. 19. Adaptive CSMA/CA with 5 nodes, packet probability = 0.01 and ϵ = 0.01

Here we can notice how, by increasing the probability of generating a packet, we still manage to send more packets than before. The protocol can resist much more to the increase of packets per node, but still fails at the extremes: if there are too many packets, the congestion can't be solved anymore, if there are too few (like the first case), there might still be collisions because there are too few examples to learn from and the nodes can't adapt well to the environment. Finally, we consider varying the exploration probability ϵ:

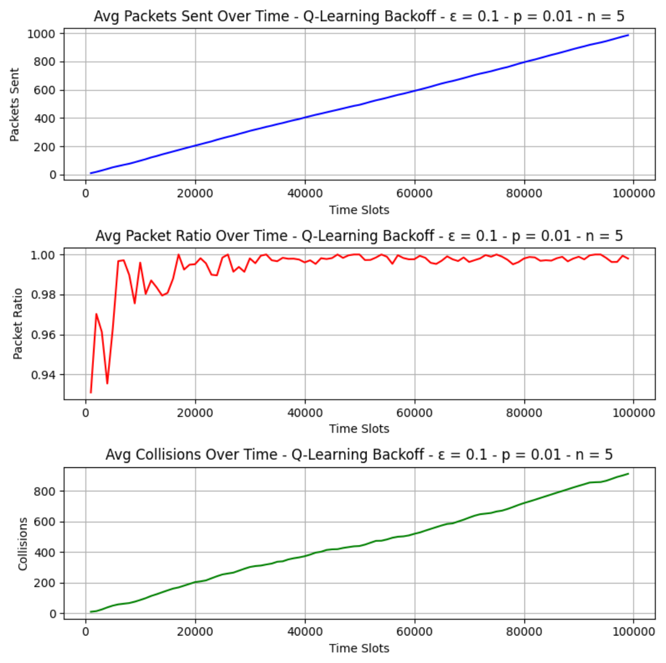


Fig. 20. Adaptive CSMA/CA with 5 nodes, packet probability = 0.01 and ϵ = 0.1

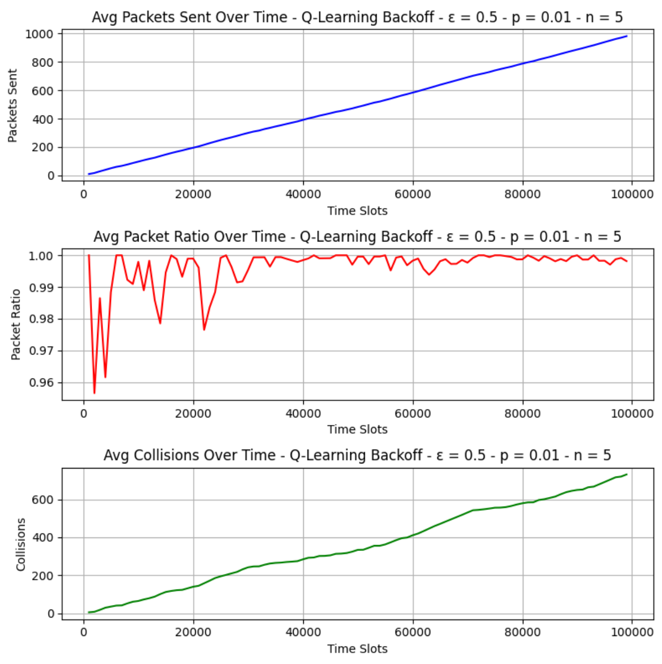


Fig. 21. Adaptive CSMA/CA with 5 nodes, packet probability = 0.01 and ϵ = 0.5

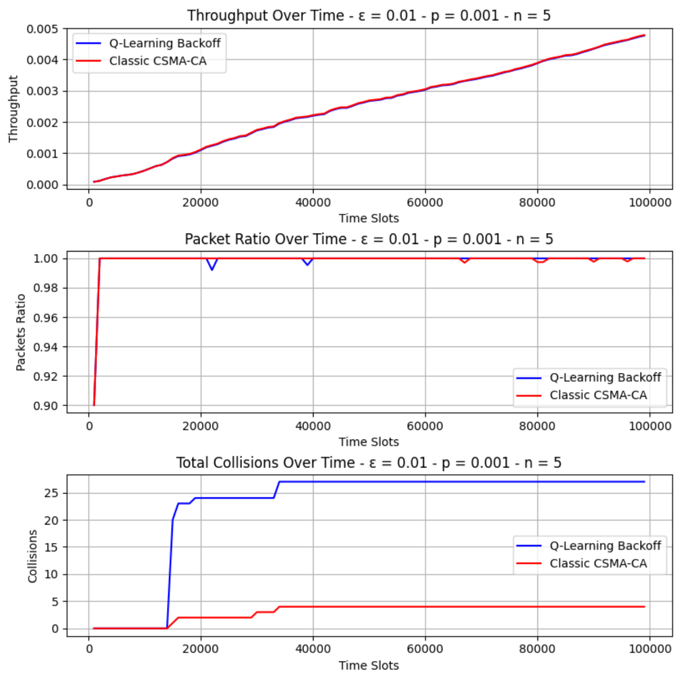


Fig. 22. Comparison with 5 nodes, packet probability = 0.001 and ϵ = 0.5

The higher ϵ is and the more we explore, indeed we can notice this effect with the packet ratio, that oscillates more when ϵ is increased.

IV. Comparison of performances

Now that we have analyzed how the performance of the two protocols varies when we change their parameters, we can fix some parameters and compare the two approaches. From now on we will consider ϵ of 0.01. The metrics used are:

- **Throughput** Amount of useful data transmitted per time unit.
- **Packet delivery ratio** The number of packets that have arrived successfully over the total number of packets generated.
- **Collisions** The number of collisions that occurred in the simulation.

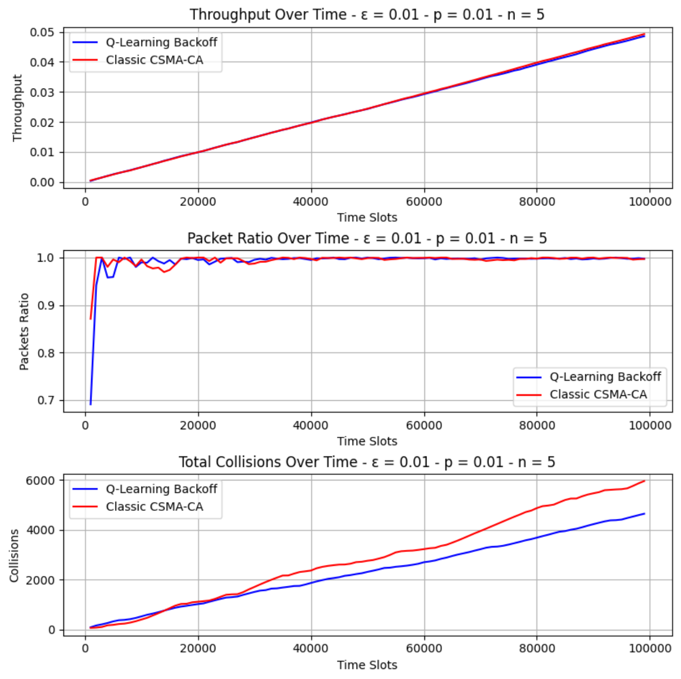


Fig. 23. Comparison with 5 nodes, packet probability = 0.01 and ϵ = 0.5

With 5 nodes and packet probability generation of 0.01 and 0.001 the overall performance is the same. We can however notice how in the case of p = 0.001, the Adaptive approach has an initial phase of collisions and then a steady one without any additional one. This is given to the exploration phase: once the algorithm found the right actions to perform, there are no more collisions.

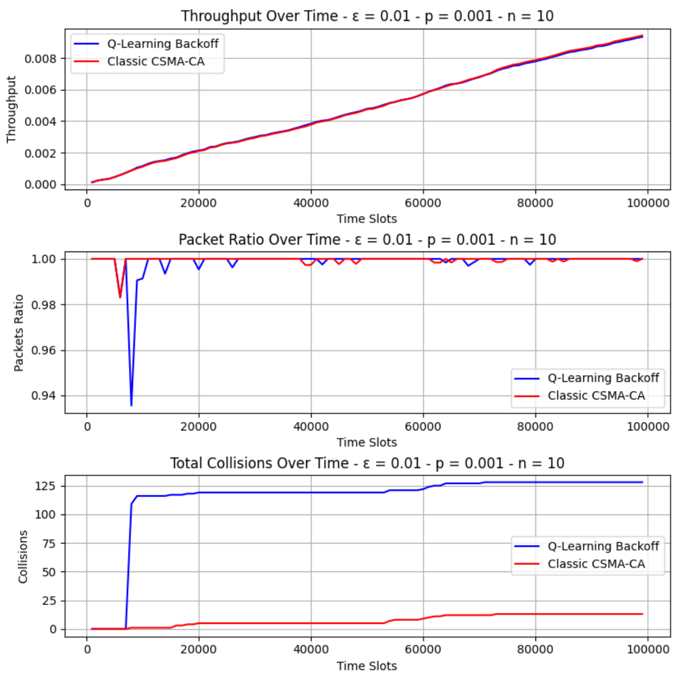


Fig. 24. Comparison with 10 nodes, packet probability = 0.001 and ϵ = 0.5

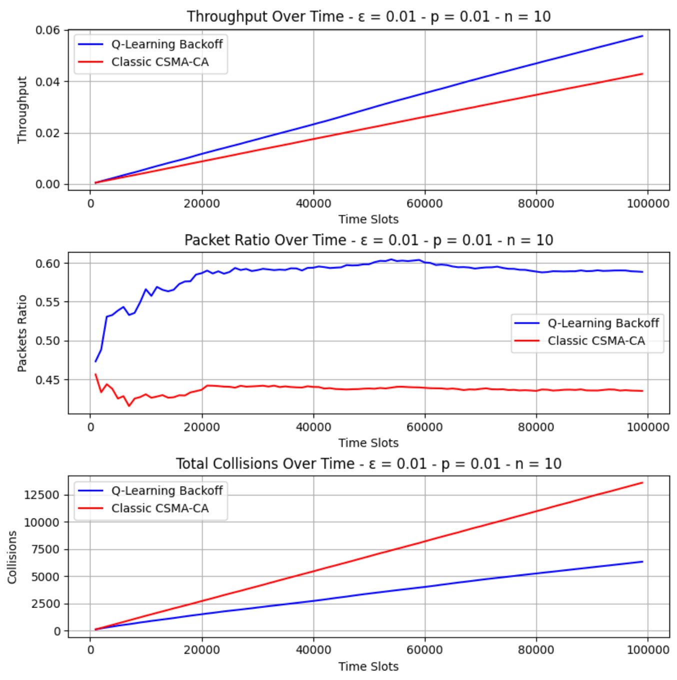


Fig. 25. Comparison with 10 nodes, packet probability = 0.01 and ϵ = 0.5

With 10 nodes instead, with a low probability we still can detect the exploration phase on the collisions, but with a higher probability, the performance improves both in all the metrics.

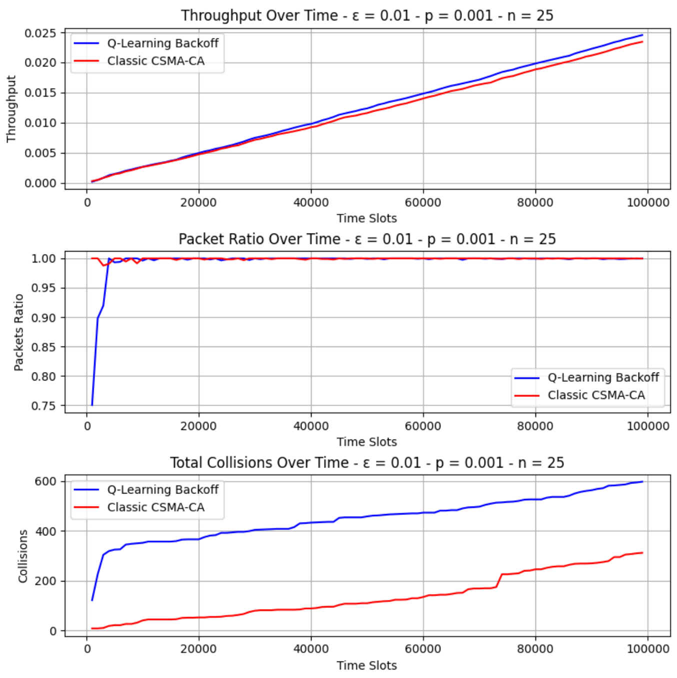


Fig. 26. Comparison with 25 nodes, packet probability = 0.001 and ϵ = 0.5

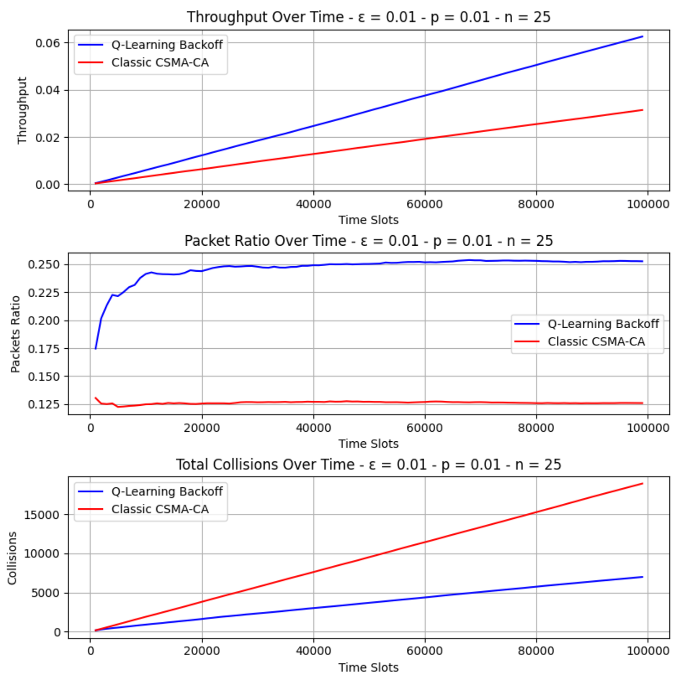


Fig. 27. Comparison with 25 nodes, packet probability = 0.01 and ϵ = 0.5

The same pattern can be seen with 25 nodes, but this time, the improvement is even greater with probability of 0.01, where the collisions are reduced by a third and the packet ratio and throughput are doubled.

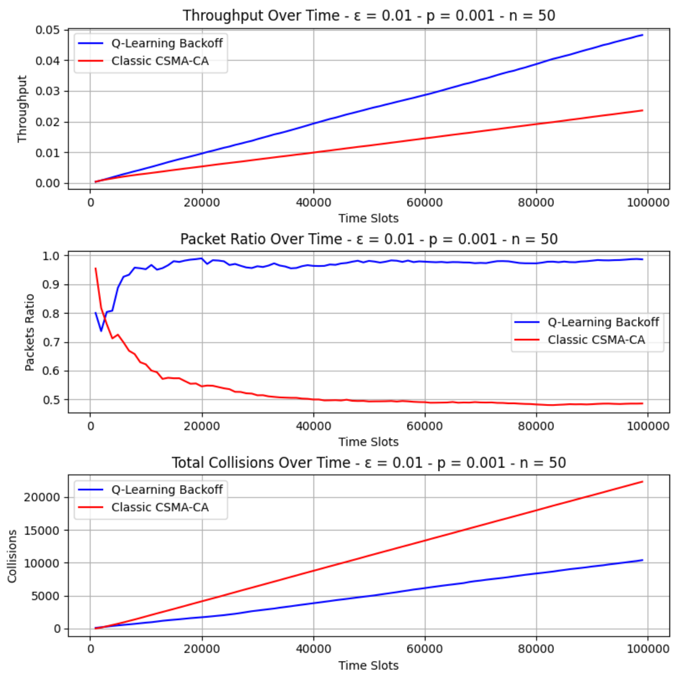


Fig. 28. Comparison with 50 nodes, packet probability = 0.001 and ϵ = 0.5

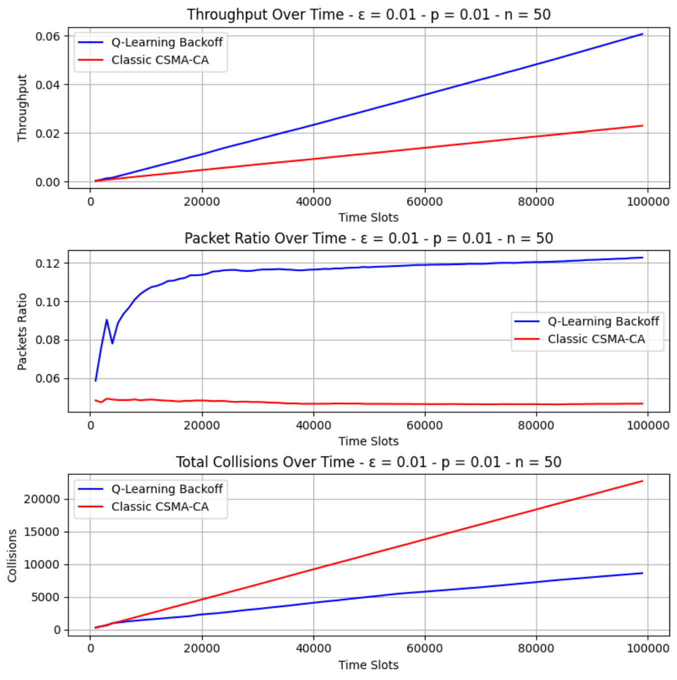


Fig. 29. Comparison with 50 nodes, packet probability = 0.01 and ϵ = 0.5

When we run the simulation with 50 nodes we can clearly see the advantage of using the adaptive method over the classical one: throughput and packet ratio are doubled and collisions are halved independently of the probability. Moreover, we can see how with a low probability, the Adaptive approach manages to send all the packets in the queues of all the nodes (the packet ratio is constantly at 1), while in the Classic CSMA/CA, on average nodes don't manage to send half of the packets in queue.

V. Conclusions

From the simulations that were conducted, we can see the improvement that the Adaptive CSMA/CA brings. When multiple nodes are transmitting, the Reinforcement Learning approach outperforms the classical one by double the amount of throughput and packet delivery ratio. This is because now each node learns how to deal with collisions, so they are able to coordinate; even in extreme scenarios, such as having 50 nodes communicating at the same time, the nodes are able to empty their queues and not discarding any packet.

References

[1] https://github.com/slashm4n/CSMA-CA/blob/main/CSMA-CA_v11%20(Classic).py
[2] https://github.com/slashm4n/CSMA-CA/blob/main/CSMA-CA_v14%20(Adaptive).py
[3] https://github.com/slashm4n/CSMA-CA/blob/main/plot_statistics.py