Project Autonomous Networking: CSMA/CA

Michele Saraceno - 1905065
January 21, 2026

Abstract

This report presents a simulation of a simplified version of the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol, implemented in `C++`. The goal of the project is to investigate how reinforcement learning techniques can be integrated into the protocol to enable adaptive channel access and improve network performance. The simulation models multiple nodes contending for the medium, and evaluates how learning-based adaptations influence collision rates, throughput, and fairness compared to the Baseline CSMA/CA behavior.

1 Description of the simulation model

This section describes the simulation model developed for this project. It presents the main assumptions, design choices, and simplifications adopted to model the CSMA/CA protocol and the reinforcement learning approach.

1.1 Assumptions

The CSMA/CA mechanism implemented in the simulation is based on the access method defined in the IEEE 802.11–2024 standard [2], as illustrated in Figure 1. The interframe spacing considered is the **DIFS**. All stations operate on a single shared channel, and RTS/CTS mechanisms are not included.
The simulation evolves in discrete time steps referred to as **ticks**, each corresponding to approximately $10\,\mu s$. During each tick, all entities in the simulation perform their actions, including carrier sensing, DIFS and backoff countdown, and packet transmission. Table 1 provides an approximate conversion between real time durations and the corresponding number of ticks.

Time slot	Duration	Ticks
DIFS	$34\mu s$	3
CWmin	$8\mu s$	1
CWmax	$4096\mu s$	410

Table 1: Conversion to ticks.

To process the data obtained from the simulation and derive performance metrics such as **throughput** and **Packet Delivery Ratio**, a physical layer data rate of 6 Mbps is assumed.

1.2 Environment

The simulation environment manages the agents and the shared communication channel. It advances the system through successive ticks and coordinates the execution of actions by all entities. In addition, the environment collects and processes the data generated during the simulation in order to evaluate its performance.

1.3 Channel

The model includes a single shared channel responsible for handling packet transmissions from the stations. If more than one station attempts to transmit during the same tick, a collision occurs: no packet is successfully transmitted, and all involved stations increase their contention window accordingly.

1.4 Agents

The agents, also referred to as **stations**, are responsible for generating and transmitting packets over the shared channel.
Each station is characterized by a unique identifier, the transmission duration of its packet, a DIFS counter, a backoff counter, and a contention window value.
At each tick, if a station does not currently have a packet to transmit, it may generate one with a small probability. When a packet is available, the station follows the channel access procedure shown in Figure 1. During a tick, a station may either decrement its DIFS or backoff counter, or attempt a transmission if the access conditions are met.

1.5 Multi-Armed Bandit

Multi-Armed Bandit is a framework for algorithms that make decision over time under uncertainty. In the version used in this project, an algorithm has K possible actions (the contention window levels) to choose from (called arms), and T rounds [4]. In each round, the algorithm chooses a contention window and collects rewards from the result of the transmission, as shown in table 2.

Event	Reward
Success	$20 +$ action index
Collision	$-10 \cdot (K -$ action index$)$

Table 2: Rewards based on the outcome of the transmission. The action index is a bias that lets the model prefer larger contention windows.

The incremental mean has been used to update the action values:

$$Q_{new} = Q_{old} + \frac{1}{n}(\text{reward} - Q_{old}) \tag{1}$$

Two decision strategies have been tested in this project: ε-greedy and UCB1.

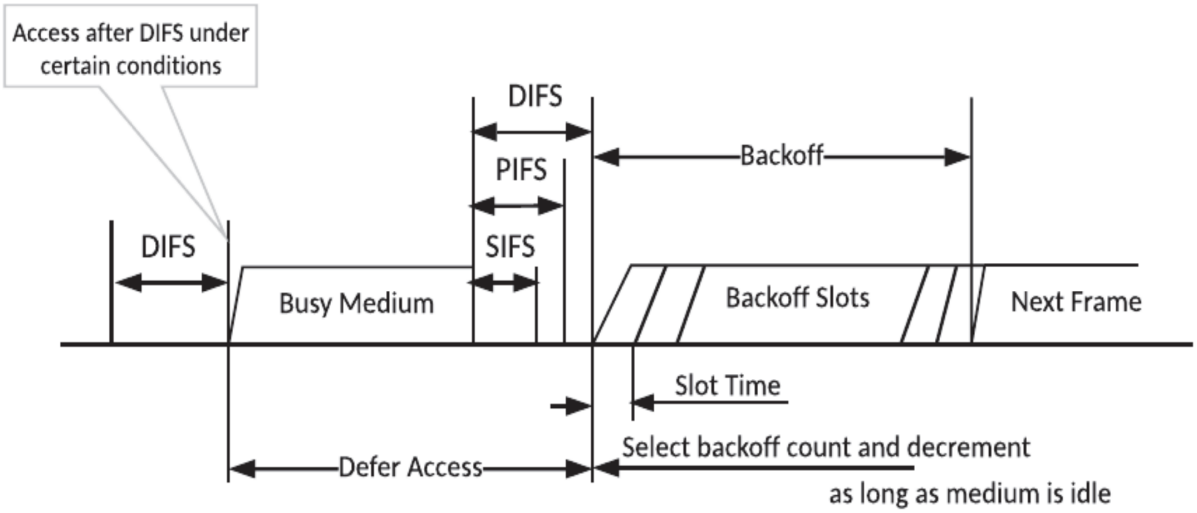

Figure 1: Basic access method.

1.5.1 ε-greedy

To decide which contention window to select, the algorithm uses a small number ε and defines the selection logic as:

$$A_t = \begin{cases} \text{argmax}_a \, Q_t(a) & \text{with probability } 1-\varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases} \tag{2}$$

With ε set as 0.1.

1.5.2 Upper Confidence Bound (UCB1)

Unlike the ε-greedy strategy, UCB1 uses a detrministic approach, calculating a *score* for each contention window level i at a certain time [3].

$$A_t = argmax_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right] \tag{3}$$

Where $Q_t(a)$ is the average reward, t is the total number that an arm is used, $N_t(a)$ is the number of times the arm a has been used, and c is the *exploration constant* (the lower, the greedier) set to 2.

1.6 Q-Learning

Q-Learning is a form of model-free reinforcement learning. It can also be viewed as a method of asynchronous dynamic programming. It provides agents with the capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains [1].
The agent-environment interaction is modeled as a Markov Decision Process defined by the tuple (S, A, R, γ), where S is the set of states, A the set of actions, R the reward function, and γ the *discount factor* (how much an agent cares about long term or immediate gratification). Each agent will keep a matrix, called *Q-table*, $Q(s, a)$ representing the action value for taking action a in state s.
The Q-table is initialized with a preference for larger contention windows, preventing initial collisions that would saturate the channel before the learning can stabilize. The decision strategy follows the Boltzmann distribution, giving priority to actions with an higher action value.
The update of the action value follows the Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \max_{a'} Q(s',a') - Q(s,a) \right] \tag{4}$$

Here $Q(s', a')$ refers to the next state. The value α is the *learning rate*, which can be fixed or dynamic (drops when the agent finds an optimal window). In case of fixed α it's value is set to 0.1, while in dynamic α its value is calculated as follows.

$$\alpha = \frac{1}{1 + N_{visits}} \tag{5}$$

Where N_{visits} is the number of times the agent has been in this specific state and performed this specific action.

2 Metrics

This section describes the metrics used to analyze the performance of the CSMA/CA protocol across five different operating modes: Baseline, Multi-Armed Bandit with ε-greedy decision strategy, Multi-Armed Bandit with UCB1 decision strategy, Q-Learning with fixed learning rate, and Q-Learning with dynamic learning rate. Throughput and Packet Delivery Ratio (PDR) are used to evaluate the scalability and efficiency of these strategies.

2.1 Throughput

Throughput represents the rate of successful data delivery. In the current implementation, a single tick corresponds to a duration of $T_{tick} = 10\mu$s. The throughput S in Mbps is calculated as the total bits successfully transmitted (BT) divided by the total simulation time:

$$S = \frac{BT}{Ticks \times 10 \cdot 10^{-6}} \times 10^{-6} \text{ Mbps} \tag{6}$$

The total bits transferred (BT) is derived from the accumulated duration of successful transmissions. Given a physical layer (PHY) rate of 6 Mbps, the bits transferred during a successful transmission of duration D (in ticks) is:

$$BT = \sum (D \times T_{tick} \times 6 \cdot 10^6) \tag{7}$$

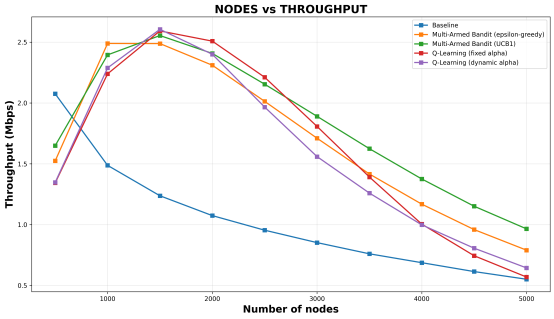

Figure 2: Throughput (Mbps) with an increase of number of nodes from 500 to 5000.

2.2 Packet Delivery Ratio (PDR)

PDR is the primary metric for assessing the efficiency of the contention resolution algorithm. It measures the probability that a transmission attempt results in a success rather than a collision. It is defined as:

$$PDR = \frac{N_{success}}{N_{success} + N_{collision}} \times 100\% \tag{8}$$

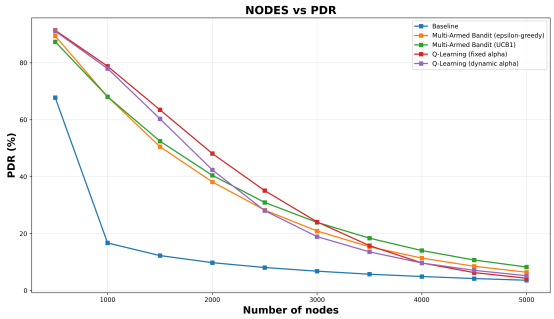

Figure 3: Packet Delivery Ratio (%) with an increase of number of nodes from 500 to 5000.

3 Results

The results are obtained by simulating a network with variable population (number of nodes), probability to generate packets, and ε value (for ε-greedy strategy). Each configuration is executed for 500000 ticks and repreated three times to report average values, minimizing the impact of variance in packet generation and backoff selection. In the following sections, the performance of the simulation has been computed with the variation of a specific parameter.

3.1 Number of nodes

The network has been tested on an increasing number of nodes each time. Starting from 500 nodes, they increased by 500 for each simulation until 5000. The results show that there is a substantial difference between the Baseline and reinforcement learning strategies. The Baseline suffers from a rapid *congestion collapse*, with the PDR dropping to 9.8% at 2000 nodes, and the throughput in constant decline. In contrast, the reinforcement learning-based strategies show an higher performance even with the increase of the population, with a peak in throughput around 1500 nodes. More specifically, the Q-Learning with fixed α emerged as the most reliable with an average efficiency almost the triple of the Baseline's, while the Multi-Armed Bandit UCB1 is slightly better at really high node densities.

3.2 Probability to Generate a Packet

At each simulation tick, if a node has no packet to send, it attempts to generate one based on a specific probability. While the other simulations utilized a generation probability of 0.0002 (0.02%), these tests increased the probability range from 0.0002 to 1.0. The results reveal a clear congestion point around 0.001. Beyond this threshold, increasing the traffic load does not improve throughput, indicating that the channel has reached full congestion. Q-Learning strategies outperform all other methods, maintaining a Packet Delivery Ratio of 60% even as traffic load increases.

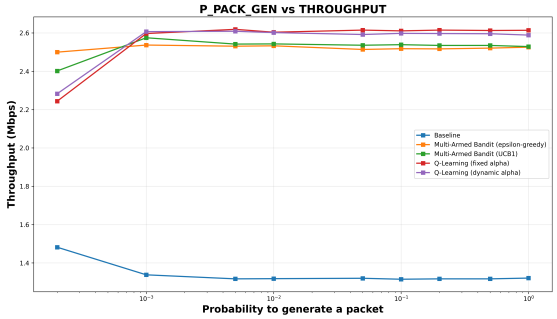

Figure 4: Throughput (Mbps) with an increase of the probability to generate a packet from 0.0002 to 1.0.

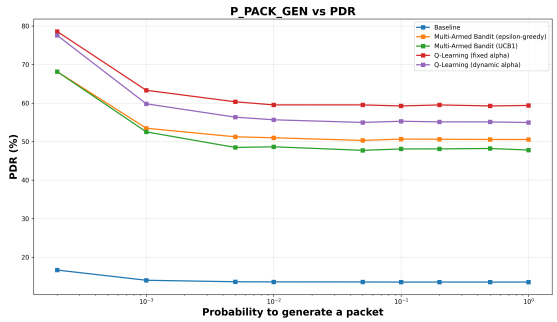

Figure 5: Packet Delivery Ratio (%) with an increase of the probability to generate a packet from 0.0002 to 1.0.

3.3 ε Value

In ε-greedy decision strategy Multi-Armed Bandit, the value of ε is used to calculate the probability to choose or not a certain arm. Normally, in the simulations the value of ε is set to 0.1 but, as shown in figure 7, different values have been tested, up to 1.0. At low exploration rates (around 0.1), the system prioritizes reliability with a peak of 68.5% Packet Delivery Ratio. As ε increases toward 0.6, throughput improves to a peak of 2.56, suggesting that randomness helps desynchronize nodes that would collide. Beyond this, the system experiences a decline in both metrics; when it reaches 1.0, the algorithm becomes completely random, causing total transmission attempts to nearly double with a really low success rate.

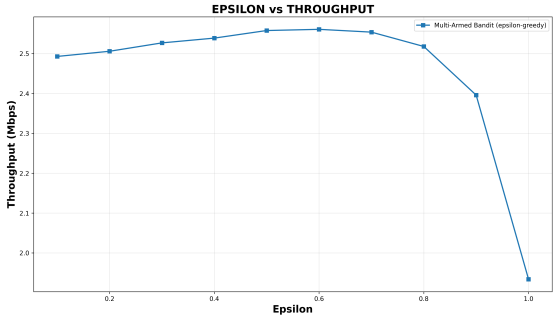

Figure 6: Throughput (Mbps) with an increase of the ε value from 0.1 to 1.0.

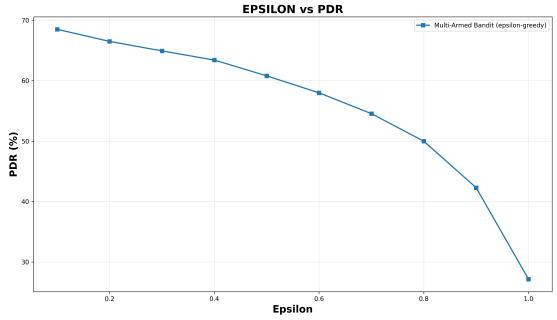

Figure 7: Packet Delivery Ratio (%) with an increase of the ε value from 0.1 to 1.0.

4 Conclusions

In conclusion, from the simulations it turns out that reinforcement learning-based strategies efficiently reduce collisions from the Baseline model by 50% for Multi-Armed Bandit and by 65% for Q-Learning. Reinforcement Learning also improves throughput in relation to the Baseline model, with peaks of 2.6 Mbps with Q-Learning, while Multi-Armed Bandit achieves the best throughput at really high population densities.
In general, this is a simple simulation of a complex protocol. It lacks RTS/CTS mechanisms and uses a single channel for contention, which is not an accurate representation of reality. In this way, it avoids dealing with real-world problems, such as hidden/exposed terminals. However, this is a good starting ground for understanding how the protocol works, how the nodes interact with the channel, how Reinforcement Learning algorithms are implemented, and how they can improve the performance of the protocol.

References

[1] Christopher J. C. H. Watkins Peter Dayan. "Q-learning". In: *Machine Learning 8* (1992). DOI: https://doi.org/10.1007/BF00992698.

[2] "IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications". In: *IEEE Std 802.11-2024 (Revision of IEEE Std 802.11-2020)* (2025), pp. 1–5956. DOI: 10.1109/IEEESTD.2025.10979691.

[3] Nicolò Cesa-Bianchi Paul Fischer Peter Auer. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning 47* (2002). DOI: https://doi.org/10.1023/A:1013689704352.

[4] Aleksandrs Slivkins. *Introduction to Multi-Armed Bandits*. 2024. arXiv: 1904.07272 [cs.LG]. URL: https://arxiv.org/abs/1904.07272.