CSMA/CA Backoff Algorithms: A GUI-driven Simulation and Analysis

by George Mensah, Tilak Marupilla, Robert Quainoo, and Soniya Kadam
Wireless and Network Engineering Program, Northeastern University, Boston, MA - USA

***Abstract*— The role of the back-off mechanism in Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocols is critical in managing network contention and optimizing channel access. This project explores various back-off mechanisms and their impact on contention window size within CSMA/CA networks. Through the development of a graphical user interface (GUI) on the Google Colab platform using Python, simulations of CSMA/CA are conducted, incorporating various backoff algorithms to determine specific use cases. The study focuses on three distinct backoff algorithms: Binary Exponential Backoff (BEB), Exponential Increase Exponential Decrease (EIED), and Logarithmic Increment Back-off. By simulating these algorithms within the context of CSMA/CA networks, this project provides insights into their effectiveness, adaptability, and overall influence on network performance. Through comprehensive analysis, the project aims to enhance understanding of backoff mechanisms in CSMA/CA protocols and their implications for network efficiency and reliability.**

Keywords*— *CSMA/CA, Backoff Mechanism, Contention Window, Binary Exponential Backoff (BEB), Exponential Increase Exponential Decrease (EIED), Logarithmic Increment Backoff, Simulation, Graphical User Interface (GUI), Performance Evaluation

I. Introduction

Carrier-sense multiple-access with collision avoidance (CSMA/CA) protocols rely on the random deferment of packet transmissions for the efficient use of a shared wireless channel among many nodes in a network. CSMA/CA protocol depends extensively on the backoff mechanism to manage network contention and optimize channel access; this class of medium access control (MAC) protocols is one of the most popular for wireless networks [2].

Distributed Coordination Function (DCF) is a MAC technique used in IEEE 802.11-based WLAN standards. It is a mandatory technique used to prevent collisions in wireless networks. DCF is used in areas where CSMA/CA is used. The technique involves the following steps: When a station has a frame to transmit, it waits for a random backoff time. If the channel is busy during the contention period, the station pauses its timer until the channel is clear. At the end of the backoff period, if the channel is idle, the station waits for an amount of time equal to Distributed Inter-Frame Space (DIFS) and then senses the channel again. If the channel is still clear, the station transmits an RTS (request to send) frame – at this point, the transmitting station sets its network allocation vector (NAV) to indicate the duration it expects to be occupied with the upcoming transmission. The destination station responds using a clear-to-send (CTS) frame if it is available, however before sending the CTS frame, the destination station adjusts its

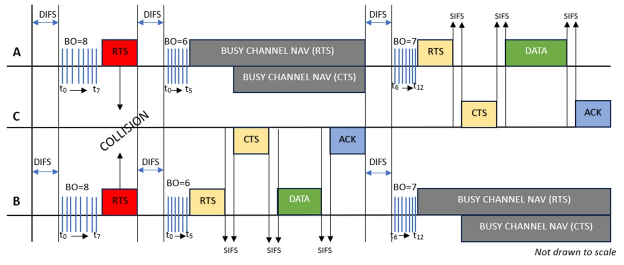


Fig. 1. Overview of CSMA/CA [1]

NAV to reflect the expected duration of the data transmission. Then the transmitting station sends the data frames. After sending the frames, the transmitting station waits for a time equal to SIFS (Short Inter-Frame Space) for acknowledgment. Finally, the station waits for the backoff time before the next transmission.

II. COMPARISON OF BACKOFF ALGORITHMS

A All three algorithms are important back-off mechanisms utilized for dealing with the network congestion scenario. The choice of a back-off algorithm plays a crucial role in ensuring efficient and fair access to the shared communication medium allowing network designers to choose the most suitable algorithm based on specific deployment scenarios and performance requirements. We attempt to compare the three backoff algorithms: Binary Exponential Backoff (BEB), Logarithmic Backoff (LB), and Exponential Increase Exponential Decrease Backoff (EIED). To do this, we set up simulations with the following constants:

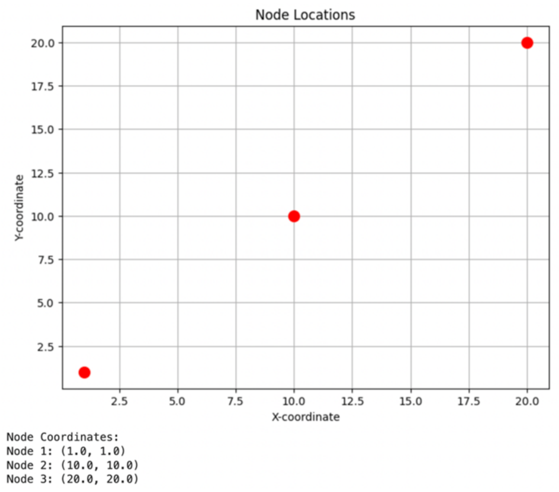
A]Distributed Inter-Frame Space (in seconds) (DIFS) = 0.05
B]Short Inter-Frame Space (in seconds) (SIFS) = 0.01
C]Slot time (in seconds) = 0.02
D]Minimum contention window size (CWmin) = 15
E]Maximum contention window size (CWmax) = 1023
F]Maximum number of retransmission attempts = 7

A. *Simulation Environment Setup*

In our simulation with 3 nodes (Node 1, Node 2, Node 3), Node 1 and Node 3 are transmitters, and Node 2 is the receiver. The node setup is displayed in Figure 2.

III. CSMA/CA with Binary Exponential Backoff (BEB)

In Binary Exponential Backoff (BEB), the contention window size is doubled after every unsuccessful attempt. In the



```
Node Coordinates:
Node 1: (1.0, 1.0)
Node 2: (10.0, 10.0)
Node 3: (20.0, 20.0)
```

Fig. 2. Simulation Environment showing node locations

initial attempt, the CWmin is used and the BEB selects a random slot from the next contention window (which is equal to the CWmin). In successive attempts, the contention window size will keep doubling for every collision till it reaches its maximum CWmax or there is a successful transfer. This is emulated in the formula: CW = min (2*CW, CWmax) [3] The actual backoff value is randomly chosen from the range [0, CW-1] where CW is the current contention window size.

A. *Analysis of the BEB codebase Architecture*

The 'update energy consumption method' calculates the energy consumed by a node based on its activity (sleep, idle, back-off, RTS, CTS, SIFS, DIFS, distance, and transmission). The energy consumption is updated accordingly.

The 'simulate transmission method' simulates the transmission process for each transmitting node. It checks if all receiving nodes are available (not already received an RTS) and initiates the backoff process before sending an RTS. If a receiving node is within the transmission range and has not already received an RTS, the transmitting node sends an RTS. The receiving node responds with a CTS if it is available. After a successful RTS/CTS exchange, the transmitting node sends the data and waits for an acknowledgment. If a collision occurs or the maximum number of retries is reached, the transmission fails. [6] The simulation also uses threading to simulate concurrent transmissions from multiple nodes. Each transmitting node runs in a separate thread, allowing for simultaneous transmission attempts.

When a node needs to transmit data, it enters the backoff process before sending an RTS (Request to Send) packet. The backoff process aims to reduce collisions by spreading out the transmission attempts over time. The node selects a random number of backoff slots from the range [0, contention window], where the contention window represents the current size of the contention window. The selection of backoff slots is performed using the line backoff slots = random.randint(0, self.contention window). Once the number of backoff slots is determined, the actual backoff time is calculated by multiplying the number of slots by the slot time (slot time). This calculation is done using the formula backoff time = backoff slots* SLOT TIME. The resulting backoff time represents the duration the node will wait before attempting to transmit. By multiplying the number of slots by the slot time, the code ensures that the backoff time is proportional to the number of slots selected. This calculation allows for a variable backoff period based on the contention window size.

After calculating the backoff time, the node enters the backoff state and waits for the calculated duration. During this time, the node defers its transmission attempt, allowing other nodes to access the medium. The backoff time calculation helps in reducing the probability of collisions by distributing the transmission attempts over different time slots. If the transmission attempt is successful, indicating there are no collisions and the node receives an acknowledgment, the contention window size is reset to its minimum value (CWmax). This allows the node to start with a smaller backoff range for the next transmission, assuming the network conditions are favorable. However, if the transmission attempt fails due to a collision or lack of acknowledgment, the contention window size is doubled. The doubling of the contention window size is performed using the line self.contention window = min(self.contention window * 2, CWmax). By doubling the contention window size, the range of backoff slots increases exponentially. This exponential increase in the backoff range helps to further spread out the transmission attempts and reduce the chances of recurring collisions. The contention window doubling process continues with each failed transmission attempt until the contention window size reaches the maximum value defined by CWmax. This limit prevents the contention window from growing indefinitely, which could lead to excessive backoff times and reduced network efficiency. [6]

B. *Simulation Setup*

1) Projected Calculations: The slot time keeps increasing as the number of slots increases. Beginning with an initial contention window size of 15 slots, subsequent attempts witness exponential growth, doubling after each collision, up to a maximum threshold defined by CWmax. For instance, after the second collision, the contention window increases by CW = min (2 * CWmin, CWmax) = min (2 * 15, 1023) = min (30, 1023) = 30 slots. The slot time also increases by 30 * 0.02 = 0.6 seconds. After the seventh unsuccessful transmission attempt, the contention window size reaches CW = min (2 * CWmin, CWmax) = min (2 * 480, 1023) = min (960, 1023) = 960 slots. However, upon a successful transmission, the contention window size resets to its minimum value of 15 slots, initiating a fresh cycle of contention. Each transmission attempt is associated with a specific time interval, calculated based on the number of slots and the given slot time of 0.02 seconds, ensuring a coordinated and adaptive approach to channel access within the network. Thus, in our setup, assuming the successful transfer of the packet occurs in the 7th attempt we observe that:

Attempt	CWmin before (slots)	Slot Time (s)	CWmin After
1 (collision)	15	0.3	30
2 (collision)	30	0.6	60
3 (collision)	60	1.2	120
4 (collision)	120	2.4	240
5 (collision)	240	4.8	480
6 (collision)	480	9.6	960
7 (success)	960	19.2	15

The process is displayed in Figure 3

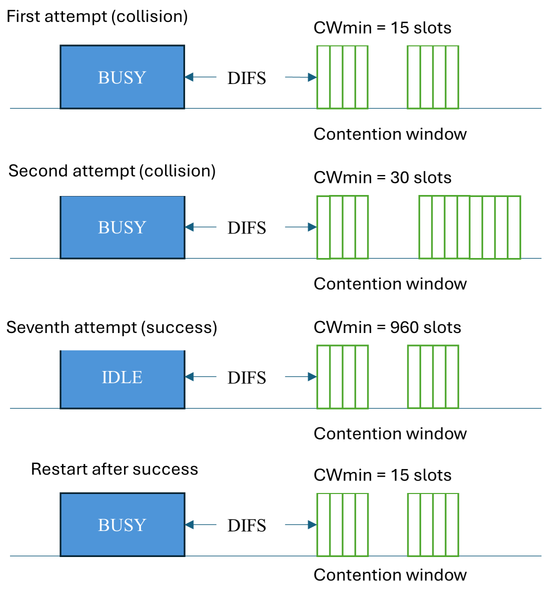


Fig. 3. BEB process

C. Simulation Output for BEB

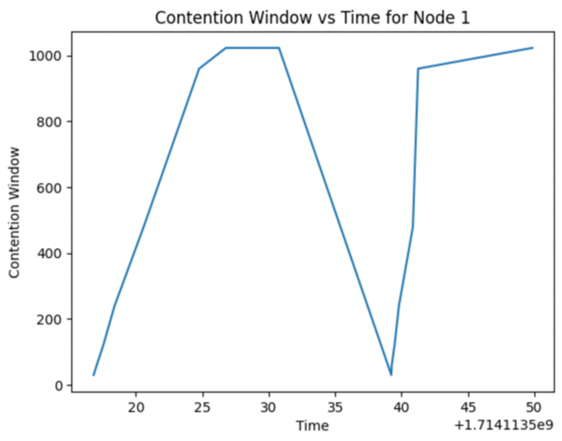


Fig. 4. BEB Node 1 Simulation

The contention window for Node 1 starts around 20 and increases exponentially until about 30 seconds when it reaches its peak at approximately 1000. After 30 seconds, there is a sudden drop in the contention window size, bringing it back to below 200. This drop suggests a successful transmission occurred at that point, causing the contention window to reset to its initial value. The contention window then continues to increase exponentially again until around 48 seconds. This is shown in Figure 4.

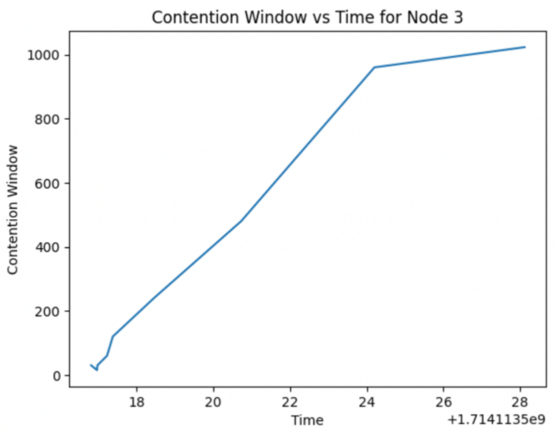


Fig. 5. BEB Node 3 Simulation

The contention window for Node 3 starts at about 20 or 30 and grows exponentially, reaching a value of around 1000 at approximately 24 seconds. A gradual increase is seen in the contention window after 24 seconds, followed by a peak near 28 seconds. This would potentially indicate no successful transmission. This is shown in Figure 5.

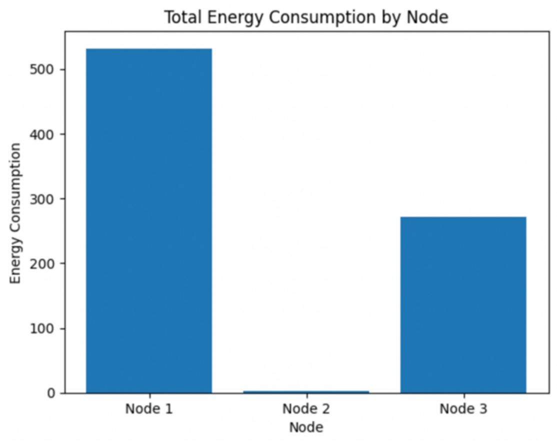


Fig. 6. BEB Total Energy Consumtion

Node 1, being a transmitter node, consumed the highest amount of energy at approximately 531.60 units.

The high energy consumption suggests that Node 1 was actively involved in transmitting data packets and spent a significant amount of time in the transmit state. The energy consumption of Node 1 is likely influenced by the number of transmission attempts, including both successful and unsuccessful attempts, as well as the duration of the backoff periods. Node 3, also a transmitter node, consumed around 270.77 units of energy, which is nearly half of Node 1's energy consumption. The lower energy consumption compared to Node 1 indicates that Node 3 had fewer transmission attempts or spent less time in the transmit state. This could be due to various factors such as a lower traffic load, shorter backoff durations, or fewer collisions experienced by Node 3 compared to Node 1. Node 2, being the receiver node, consumed significantly less energy (2 units) compared to the transmitter nodes. This is shown in Figure 6.

IV. CSMA/CA with Exponential Increment and Exponential Decrement (EIED)

In exponential increase exponential decrease (EIED) backoff algorithm, the contention window is increased and decreased by backoff factors according to collision and successful events respectively. The initial attempt involves transmitting a packet using the CWmin. If the transmission is unsuccessful (a collision occurs), the contention window is increased by a backoff factor a. If the transmission after the collision is successful, then the contention window is decreased by a backoff factor b. The EIED backoff algorithm can be represented by: CW = min (a*CW, CWmax) CW = max (CWmin/ b, CWmin) [3]

A. Analysis of the EIED codebase Architecture

When a node needs to transmit data, it enters the backoff process before sending an RTS (Request to Send) packet. The backoff process aims to reduce collisions by spreading out the transmission attempts over time. The node selects a random number of backoff slots from the range [0, contention window], where the contention window represents the current size of the contention window. The selection of backoff slots is performed using the Python-based logic: backoff slots = random.randint(0, self.contention window). Once the number of backoff slots is determined, the actual backoff time is calculated by multiplying the number of slots by the slot time. This calculation is done using the formula backoff time = backoff slots * slot time. The resulting backoff time represents the duration the node will wait before attempting to transmit. After calculating the backoff time, the node enters the backoff state and waits for the calculated duration. During this time, the node defers its transmission attempt, allowing other nodes to access the medium. [8] If the transmission attempt is successful, meaning there are no collisions and the node receives an acknowledgment, the contention window size is decreased using the EIED algorithm. The decrease is performed using the formula in Python as self.contention window = max(int(self.contention window *Beta), CWmin, where BETA is a constant value between 0 and 1. This exponential decrease allows the node to reduce its backoff time for the next transmission, assuming the network conditions are favorable. [7] On the other hand, if the transmission attempt fails due to a collision or lack of acknowledgment, the contention window size is increased using the EIED algorithm. [7] The increase is performed using the formula in Python: self*contention window = min(int(self. contention window*alpha), CW MAX), where ALPHA is a constant value greater than 1. This exponential increase in the contention window size helps to spread out the transmission attempts and reduce the chances of recurring collisions.

B. Simulation Setup

1) Projected Calculations: In EIED, the contention window size (CW) undergoes dynamic adjustments based on the outcome of transmission attempts. It increases the contention window size upon failed transmissions till it reaches the maximum contention window and reduces the contention window upon a successful transmission till it reaches the minimum contention window. In our setup, we assigned a=1.25 and b=0.8.

Attempt	CWmin before (slots)	Slot Time (s)	CWmin After
1 (collision)	15	0.3	19
2 (collision)	19	0.38	24
3 (collision)	24	0.48	30
4 (collision)	30	0.6	38
5 (collision)	38	0.76	48
6 (collision)	48	0.96	60
7 (success)	60	0.96	48

Upon a failed transmission attempt, the contention window size is increased by a factor of 1.25, gradually expanding to accommodate potential congestion and mitigate collisions. Conversely, when a transmission succeeds, the contention window size undergoes a reduction, decreased by 0.8, to promote efficient channel utilization and mitigate unnecessary delays. For instance, starting with an initial contention window size of 15 slots, it reaches CW = CWmin * 1.25 = 15 * 1.25 = 18.75 (rounded to 19) slots and reaches 60 slots after the sixth unsuccessful transmission. However, upon a successful transmission, the contention window size is revised back to 48 slots. These adjustments influence the duration of each transmission attempt, with the calculated number of slots translating to specific time intervals, ensuring a balance between contention window size and transmission efficiency. This entire process is shown in Figure 7.

C. Simulation Output for EIED

Node 1's contention window graph shows a sawtooth-like pattern. The contention window starts low around 15-20, then rapidly increases exponentially to around 80, before sharply dropping down to the minimum value. This pattern repeats, indicating cycles of collisions (exponential increase) followed by successful transmissions (exponential decrease). The EIED mechanism is visible.

Node 3's graph shows a smoother increase in contention window over time from around 15 to 60. This suggests Node 3 experienced gradual, but fewer, collisions compared to Node 1. The exponential increase is evident, but there are no sharp drops, indicating Node 3 had less successful transmissions during the simulation time to trigger the exponential decrease.

Node 1, being a transmitter node, consumed the highest amount of energy at approximately 531.60 units.

Node 1 consumed the most energy by far at around 103 units. This aligns with its contention window pattern showing repeated collisions and back-offs which consume more energy.

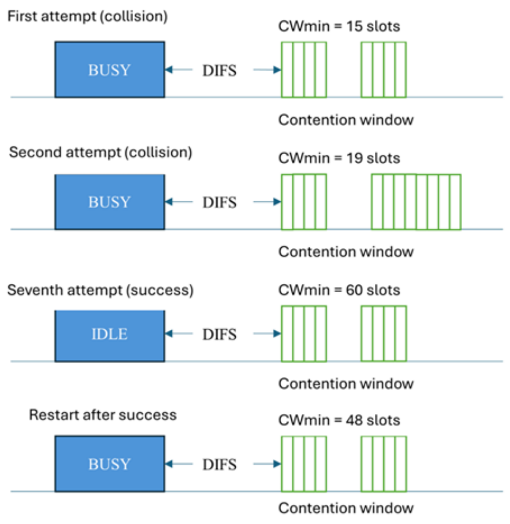


Fig. 7. EIED process

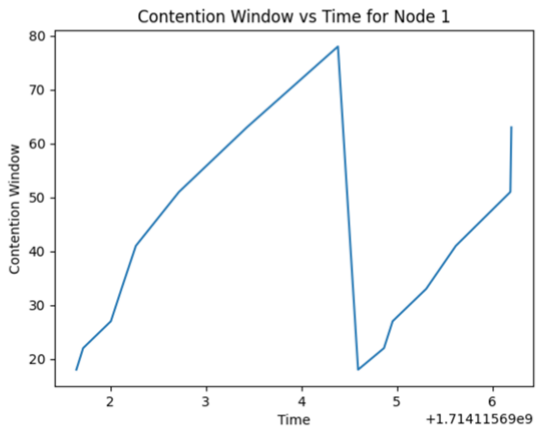


Fig. 8. EIED Node 1 Simulation

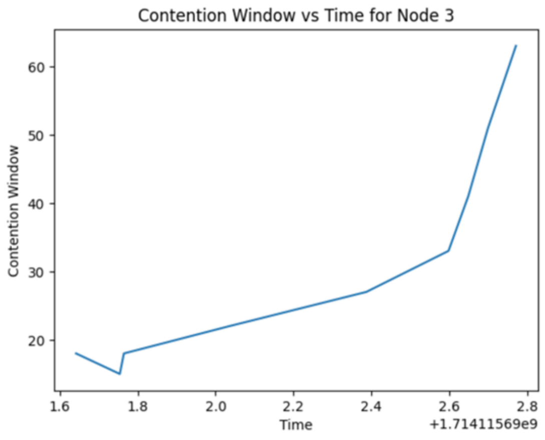


Fig. 9. EIED Node 3 Simulation

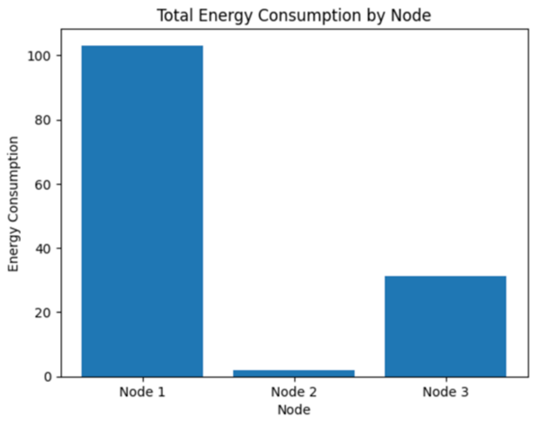


Fig. 10. EIED Total Energy Consumtion

The high contention windows indicate longer backoff times. Node 3 has the second highest energy consumption at around 31 units. While lower than Node 1, this still indicates significant energy spent in the gradually increasing backoff times as its contention window grew. Node 2 consumed very little energy at 2 units, suggesting it received successfully.

V. CSMA/CA WITH LOGARITHMIC INCREMENT BACKOFF

In logarithmic backoff, the waiting time before retransmission increases logarithmically with the number of successive collisions or failed transmission attempts. This is to gradually increase the minimum contention window size in response to congestion or contention. A basic example of a formula for calculating the contention window size CW in a logarithmic backoff algorithm is: CW = CWmin + klog2 (n) [4]

Where k is a scaling factor that determines the rate of increase of the contention window size and n is the number of consecutive unsuccessful transmission attempts (collisions). Log2 (n) is the base-2 logarithm of n. In this formula, as n increases (indicating more collisions), the contention window size CW grows logarithmically with n. The scaling factor k determines the rate of increase, allowing for adjustments to the growth rate based on specific requirements and network conditions. Logarithmic backoff is highly dependent on the protocol used to generate it. This is because the protocol determines what happens after a successful transmission of a packet.

A. Analysis of the LIB codebase Architecture

When a node needs to transmit data, it enters the backoff process before sending an RTS (Request to Send) packet. The backoff process aims to reduce collisions by spreading out the transmission attempts over time. The node calculates the contention window size using the logarithmic formula 2 ** self.retries, where self.retries represent the number of retransmission attempts. This formula ensures that the contention window size grows logarithmically with each retry. Once the contention window size is determined, the node selects a random number of backoff slots from the range [0, 2 ** self.retries - 1]. The selection of backoff slots is performed using the formula in Python written as: backoff slots = random.randint(0, 2 ** self.retries - 1). This logarithmic increase in the range of backoff slots allows for a larger spread of transmission attempts as the number of retries increases. After selecting the number of backoff slots, the actual backoff time is calculated by multiplying the number of slots by the slot time (SLOT TIME). This calculation is done using the formula backoff time = backoff slots * SLOT TIME.The resulting backoff time represents the duration the node will wait before attempting to transmit. During the backoff process, the node enters the backoff state and waits for the calculated backoff time. This waiting period allows other nodes to access the medium and helps in reducing collisions.

B. Simulation Setup

1) Projected Calculations: There are two main protocols used to define what happens to CWmin: the first is that CWmin gets reset to its base after a successful transmission (much like BEB) or CWmin gets assigned a random value. In our setup, we considered both showing that CWmin can either be 15 (original value) or 28 (random value). In a logarithmic backoff scenario, the contention window size evolves logarithmically with each unsuccessful transmission attempt, reflecting a strategic adaptation to network conditions. Commencing with a modest contention window size of 15 slots, subsequent attempts witness an incremental increase, following the logarithmic progression. For instance, after the seventh unsuccessful transmission attempt, the contention window size escalates to 41 slots, embodying the logarithmic growth pattern. Each transmission attempt is associated with a specific time interval, calculated based on the number of slots and the provided slot time of 0.02 seconds, ensuring a calibrated approach to channel access within the network. This process is shown in Figure 11.

TABLE I
ATTEMPT AND CWMIN CHANGES

Attempt	CWmin before (slots)	Slot Time (s)	CWmin After
1 (collision)	15	0.3	15
2 (collision)	15	0.3	25
3 (collision)	25	0.5	30
4 (collision)	30	0.6	35
5 (collision)	35	0.7	38
6 (collision)	38	0.76	41
7 (success)	41	0.82	15 / 28

If the transmission attempt is successful, meaning there are no collisions and the node receives an acknowledgment, the number of retries is reset to zero. This indicates that the node can start with the minimum contention window size (CWmin) for the next transmission. [9] However, if the transmission attempt fails due to a collision or lack of acknowledgment, the number of retries is incremented. The contention window size for the next backoff is then calculated using the logarithmic formula 2 *self.retries. This logarithmic increase in the contention window size ensures that the backoff time grows exponentially with each failed transmission attempt, helping to spread out the transmission attempts and reduce the chances of recurring collisions. [9]

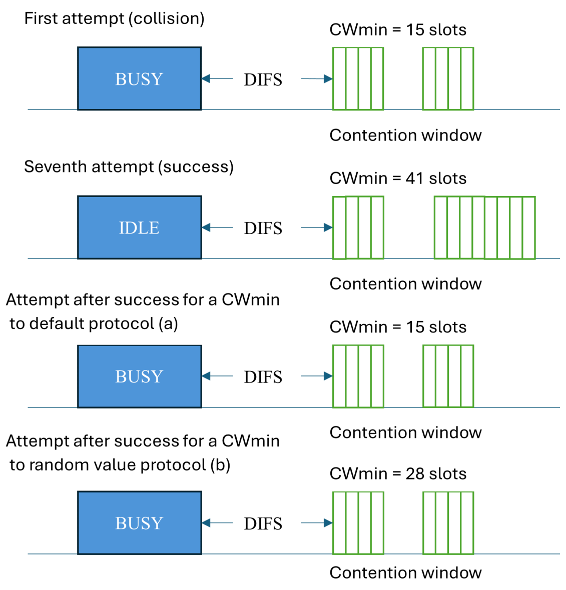

First attempt (collision)
CWmin = 15 slots
BUSY
DIFS
Contention window
Seventh attempt (success)
CWmin = 41 slots
IDLE
DIFS
Contention window
Attempt after success for a CWmin to default protocol (a)
CWmin = 15 slots
BUSY
DIFS
Contention window
Attempt after success for a CWmin to random value protocol (b)
CWmin = 28 slots
BUSY
DIFS
Contention window

Fig. 11. LIB process

C. Simulation Output for LIB

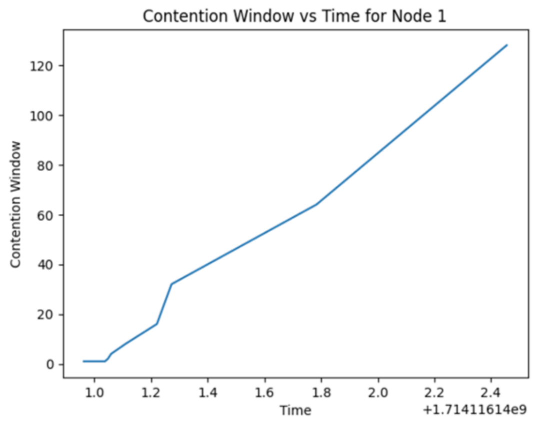


Fig. 12. LIB Node 1 Simulation

In Node 1, the contention window starts small (around 20) and increases logarithmically over time, reaching over 120 by the end. This indicates multiple backoffs and retransmission attempts due to collisions or busy medium. The logarithmic increment is visible.

Node 3, there is a single short spike in contention window to around 120 before dropping back down. This suggests Node 3 had one major backoff event before successfully transmitting.

Node 1, being a transmitter node, consumed the highest amount of energy at approximately 531.60 units.

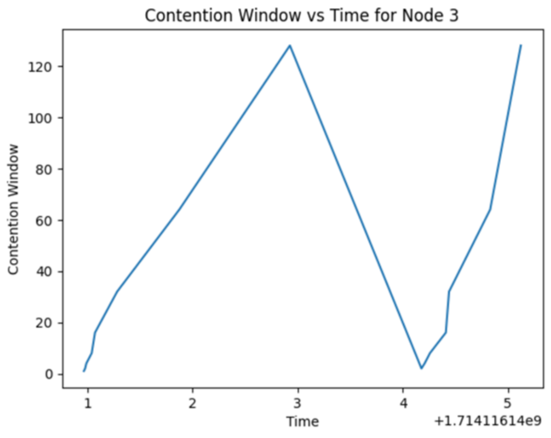


Fig. 13. LIB Node 3 Simulation

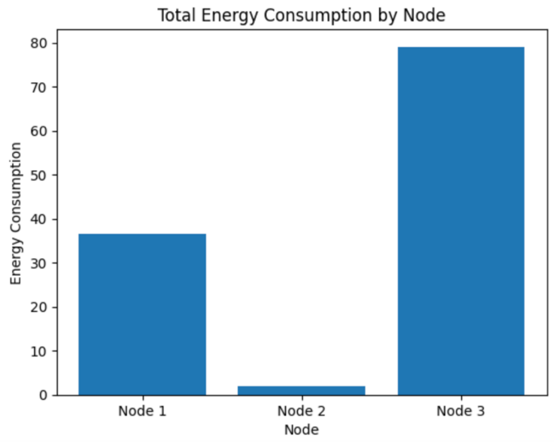


Fig. 14. LIB Total Energy consumed

Node 3 consumed the most total energy at around 79 units. This aligns with it having the highest single contention window spike, indicating significant time spent in backoff consuming energy. Node 1 consumed the second most energy at around 37 units. Its contention window grew exponentially over more time, leading to steady energy consumption. Node 2 consumed very little energy at 2 units, suggesting it received successfully.

VI. Conclusion

In this project, we examined backoff mechanisms in CSMA/CA networks using a GUI and associated backoff methods implemented on Google Colab using Python. We simulated CSMA/CA with three backoff algorithms: BEB, EIED, and Logarithmic Increment. Binary Exponential Backup quickly adapts to contention but may increase delays. EIED balances adaptation and contention window size. Logarithmic Increment offers gradual backoff. Our analysis showed each algorithm impacts performance, energy use, and fairness differently. BEB had the highest energy use, EIED was more balanced, and the Logarithmic Increment was moderate. Choosing the right algorithm is crucial based on network specifics like node count, traffic load, and desired performance metrics.

References

[1] A. M. Hamzah and Y. J. K. Nukhailawi, "The backoff in intermediate networks for a real-time system embedded ethernet," in *2022 Fifth College of Science International Conference of Recent Trends in Information Technology (CSCTIT)*, 2022, pp. 277–281.

[2] M. Cagalj, S. Ganeriwal, I. Aad, and J.-P. Hubaux, "On selfish behavior in csma/ca networks," in *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, vol. 4, 2005, pp. 2513–2524 vol. 4.

[3] J. Konorski, "CSMA/CA performance under backoff attacks: A game-theoretic context," in *MMB & PGTS 2004, 12th GI/ITG Conference on Measuring and Evaluation of Computer and Communication Systems (MMB) together with 3rd Polish-German Teletraffic Symposium (PGTS), September 12-15, 2004, Dresden, Germany*, P. Buchholz, R. Lehnert, and M. Pióro, Eds. VDE Verlag, 2004, pp. 349–358.

[4] N.-O. Song, B.-J. Kwak, J. Song, and M. Miller, "Enhancement of ieee 802.11 distributed coordination function with exponential increase exponential decrease backoff algorithm," in *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.*, vol. 4, 2003, pp. 2775–2778 vol.4.

[5] S. Manaseer, M. Ould-Khaoua, and L. Mackenzie, *On the logarithmic backoff algorithm for MAC protocol in MANETs*. IGI Global, 2008, pp. 174–184.

[6] M. Al-Hubaishi, T. Alahdal, R. Alsaqour, A. Berqia, M. Abdelhaq, and O. Alsaqour, "Enhanced binary exponential backoff algorithm for fair channel access in the ieee 802.11 medium access control protocol," *International Journal of Communication Systems*, vol. 27, no. 12, pp. 4166–4184, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.2604

[7] C. Ye, Y. Li, and A. Reznik, "Performance analysis of exponential increase exponential decrease back-off algorithm," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–6.

[8] N.-O. Song, B.-J. Kwak, and L. Miller, "Analysis of eied backoff algorithm for the ieee 802.11 dcf," in *VTC-2005-Fall. 2005 IEEE 62nd Vehicular Technology Conference, 2005.*, vol. 4, 2005, pp. 2182–2186.

[9] F. Hong-yu, X. Lei, and L. Xiao-hui, "Logarithmic backoff algorithm of mac protocol in ad hoc networks," in *2010 International Conference on E-Business and E-Government*, 2010, pp. 1695–1698.