**An Application for Information Transfer, using Ultrasound Waves**

**Authors:**

Yoni Grinberg ID: 307868257 [Yoni.Grinberg@e.braude.ac.il](mailto:Yoni.Grinberg@e.braude.ac.il)

Bar Harush ID: 205713308 Bar.Harush@e.braude.ac.il

Group:26-1-D-7

**Supervisor:**

Dr. Elena Ravve

***Abstract.***

*Standard wireless communication technologies such as Wi-Fi face significant limitations regarding signal containment, security and propagation through conductive media.*

*To address these challenges, this project presents a design and implementation of many-to-many ultrasonic data transfer network utilizing standard smartphone hardware which is capable of scalable network layer compatible with Android and IOS platforms.*

*Keywords:*

* Primary Keywords: Ultrasonic Communication, Data-over-Sound, Many-to-Many Networking
* Protocol Keywords: CSMA/CA, RTS/CTS Handshake, Collision Avoidance
* Technical/Method Keywords: Frequency-Shift Keying (FSK), FFT (Fast Fourier Transform), Signal Processing.
* Platform/Context Keywords: Cross-Platform (Android & iOS), Acoustic Data Transmission

# 

# Chapter 1. INTRODUCTION:

The core idea for this project stems from the limitations identified in current wireless communication technologies, particularly when applied to challenging physical environments where RF (Radio Frequency) fails such as Conductive Media.

The project leverages the fact that **acoustic waves (ultrasound) propagate exceptionally well through liquids and tissue**, making it the essential alternative where RF fails.

The necessity for ultrasonic data transfer extends beyond physical medium constraints. Standard RF technologies, such as Wi-Fi and Bluetooth, operate in crowded frequency spectrums that are susceptible to electromagnetic interference and signal congestion. Furthermore, RF signals easily penetrate solid objects like walls, creating significant security risks known as "snooping," where data can be intercepted from outside the intended physical space. In contrast, ultrasonic waves are naturally contained by solid barriers, providing an inherent layer of physical security and privacy.

In this project, we will address these challenges by developing a cross-platform application that enables many-to-many data transfer using ultrasonic waves. By implementing advanced collision avoidance protocols, we aim to create a reliable acoustic network that operates seamlessly across both Android and iOS devices, independent of traditional RF infrastructure.

**1.1. Organization Of The Book:**

**Chapter 1: INTRODUCTION:**

**1**.1. Organization Of The Book

1.2. What Are We Going To Do?

1.3. Why Is The Project Not Trivial?

1.4. What Has Been Done Until Now?

1.4.1. What Is Wrong With What Is Done Till Now?

1.4.2. How Do We Plan To Improve The Situation?

1.4.3. Why Do We Think That Our Improvement Works?

1.4.4. Rationale For Prioritizing CSMA/CA

1.4.5. What Do We Do If Our Improvement Does Not Work?

1.5. Difficulties In Setting This Project

1.5.1. Algorithmic Challenges

1.5.1.1. How Do We Plan To Overcome Them?

1.5.1.2. What Do We Do If Our Expectations Do Not Hold?

1.5.2. Hardware Equipment

**Chapter 2: Background And Related Work**

2.1. Reasons To Use Ultrasound

2.2. Specific Environmental Struggles

2.3. What Is Ultrasound Communication

2.4. Benefits Of Sending Data Over Sound: Summary

**Chapter 3: General Steps Of Implementation**

3.1. Data Encoding: Frequency-Shift Keying (FSK)

3.1.1. Frequency Selection

3.1.2. Bit Duration

3.2. Connection Establishment: CSMA/CA With RTS/CTS

3.2.1. RTS\CTS Frame Structure

3.2.2 Transmitter Implementation

3.2.3. Receiver Implementation

3.2.4 Pseudo Code And Algorithms

3.2.5. Collision Avoidance Algorithm (CSMA/CA) & Flow

3.3. Cross-Platform Implementation (Android vs. iOS)

**Chapter 4: Expected Results**

**Chapter 5: UML**

**Chapter 6: Testing**

6.1. Bit Packing And Fragmentation Logic

6.1.1. Checksum Calculation Accuracy

6.1.2 Packet Serialization

6.1.3 Frequency Analysis Algorithm

6.2 Physical Layer Testing

6.2.1. Bit Duration and Timing

6.3. Integration Testing

6.3.1. Single Packet Transmission

6.3.2. RTS/CTS Handshake Verification

6.4. Field Testing

6.4.1 Maximum Effective Range

6.4.2 Noise Immunity

6.4.3 Wall/Barrier Penetration Test

**Chapter 7: FR\NFR Requirements**

**Chapter 8: References**

**1.2.** **What Are We Going to Do?**

The idea thus evolved from a simple acoustic link into a full many-to-many communication protocol design challenge. The current project's rationale is to bridge the gap between a lab-based physical demonstration and a robust, scalable, and secure network layer.

The objective is to establish a secure, multi-user ultrasonic network that proves the viability of this non-RF (Radio Frequency), acoustic method for applications in environments like operating rooms, laboratories, or personal proximity systems where the benefits of liquid/tissue penetration and physical security (signal containment) are paramount. This requires introducing a sophisticated channel arbitration method like CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) and ensure seamless cross-platform compatibility between Android and the iPhone Operating System (iOS).

**1.3.** **Why Is The Project Not Trivial?**

While previous academic work successfully pioneered a basic one-to-one ultrasonic data link between two Android devices, the system lacked the required network layer for real-world utility. For any practical application (e.g., secure medical pairing, multi-party data exchange), the system must be scalable to allow multiple devices to simultaneously listen, transmit, and coordinate without data collision.

**1.4. What Has Been Done Until Now?**

This project builds upon the foundational research and implementation conducted by previous engineering students at Braude College. Specifically, we acknowledge the work of Miras Safadi, Rani Hassan Smart Visit Card (SVC) [6], Tal Zilberman, Ariela Havkin Ultra Sound Data Transfer Many To Many [7], Taimor Fares and Fares Jaraisy Inaudible Information Transfer between Android Based Devices [8].

These teams successfully established Proof of Concept (PoC) for a one-to-one ultrasonic communication channel between two Android devices. Their key achievements included:

* **Physical Layer Implementation:** These teams researched and selected **Frequency-Shift Keying (FSK)** as the optimal modulation technique for acoustic data transfer [7][1].
* **Frequency Selection:** These teams determined that the frequency range of **18 kHz to 19 kHz** was most effective for mobile speakers and microphones, remaining inaudible to humans while detectable by standard hardware [7][2].
* **Data Processing:** These teams implemented a flow using Fast Fourier Transform (FFT) to convert recorded audio from the time domain to the frequency domain to decode binary data [7][3].

**1.4.1. What Is Wrong With What Has Been Done Until Now?**

While the previous implementation succeeded in establishing a basic link, it lacked the necessary **Network Layer** and **Cross-Platform** capabilities required for real-world utility.

* **Lack of Network Protocol (Scalability):** The previous system was strictly limited to a **one-to-one** connection. It did not have a mechanism to handle multiple devices simultaneously. Without a channel arbitration protocol (like CSMA/CA), if three or more devices attempted to communicate, their signals would collide, rendering the data unreadable. It could not support a "many-to-many" topology.

**Single Platform Restriction:**

The previous project was developed exclusively for the **Android** operating system. In a real-world scenario (such as contact tracing or medical environments), users carry heterogeneous devices. A system that cannot communicate between an Android phone and an iPhone (iOS) has limited practical application.**1.4.2.** **How do we plan to improve the situation?**

Our goal is to expand the communication channel from one- to -one, to many- to- many. We will introduce a sophisticated channel arbitration method, specifically the Carrier Sense Multiple Access CSMA/C) protocol, to manage network traffic.  
We aim to bridge the gap between a lab demonstration and a robust network by ensuring seamless cross-platform compatibility between Android and iOS.

**1.4.3. Why Do We Think That Our Improvement Works?**

**A. Collision Avoidance:** Since we are utilizing the CSMA/CA protocol, the system can effectively arbitrate many-to-many traffic. This protocol allows verification of devices channel availability before transmitting, ensuring data is received reliably while preventing signal collisions and packet loss.

**B. Cross-Platform Compatibility:** Achieving seamless operation between platforms is highly viable, as robust frameworks for this purpose already exist in the market. These technologies facilitate standardized Digital Signal Processing (DSP) and allow uniform access to device hardware (microphone/speaker). Leveraging these existing capabilities ensures that developing our own cross-platform transfer solution is feasible**.**

**1.4.4. Rationale For Prioritizing CSMA/CA [7]:**

While Slotted ALOHA and Token Passing are viable alternatives, we selected CSMA/CA with RTS/CTS as the primary protocol for three engineering reasons specific to the ultrasonic smartphone environment:

1. Resilience to Dynamic Network Topology (vs. Token System) The Big Brother" application is designed for spontaneous encounters where devices constantly enter and leave the 2-meter effective range.

* **Token System Weakness:** Token protocols rely on a defined logical ring. If a user holding the "Token" walks away or their battery dies, the token is lost, causing a network stall until a complex "election" process generates a new token.
* **CSMA/CA Strength:** CSMA/CA is decentralized and asynchronous. It does not require a fixed network structure, allowing devices to join or leave the conversation instantly without crashing the network flow.

2. Asynchronous Operation (vs. Slotted ALOHA) Mobile operating systems (Android/iOS) suffer from non-deterministic latency and are not hard Real-Time Operating Systems (RTOS).

* **Slotted ALOHA Weakness:** This protocol requires strict time synchronization to ensure all devices agree on exactly when a 28ms "slot" begins. Achieving microsecond-level synchronization between heterogeneous devices (e.g., an old Android and a new iPhone) without a central server is technically prohibitive. [9]
* **CSMA/CA Strength:** CSMA/CA operates on a "Listen-Before-Talk" principle. It does not require a global clock, making it far more robust for devices with varying processing speeds and OS delays.

3. The Hidden Node Solution (RTS/CTS) In acoustic environments, the "Hidden Node Problem" is prevalent, where two emitters are out of range of each other, but both are in range of the same receiver. [6]

* **ALOHA/Token Weakness:** Pure ALOHA does not check for receiver availability, leading to unavoidable collisions at the "hidden" receiver. [9]
* **CSMA/CA Strength:** The RTS/CTS (Request to Send / Clear to Send) handshake explicitly reserves the medium at the receiver's end. This ensures that even "hidden" nodes know the channel is busy, a critical feature for reliable data transfer in our specific topology**.** [7]

**1.4.5.** **What Do We Do If Our Improvement Does Not Work?**

**A. Protocol Fallback (Collision Avoidance Failure)**

**Slotted ALOHA (Time-Slot Synchronization)** [9]

If the asynchronous nature of CSMA/CA leads to high collision rates, we will implement **Slotted ALOHA** as a fallback. While originally designed for RF networks, this protocol can be adapted "in the same way" for the ultrasonic medium.

* **Mechanism:** Time is divided into discrete intervals called "slots." Each slot corresponds to the time required to transmit a single fixed-length frame (approx. 28ms in our current design).
* **Operational Logic:**
  + Devices are synchronized (using system time or a beacon signal).
  + A device is only permitted to begin transmission at the exact beginning of a time slot.
  + By forcing alignment, "partial collisions" (where one packet overlaps the tail of another) are eliminated.
* **Collision Handling:** If two devices transmit in the same slot, a complete collision occurs. Both devices detect failure (via lack of ACK) and wait a random number of slots before retrying.
* **Advantage:** This simplifies the arbitration process compared to RTS/CTS, reducing the handshake overhead, though it introduces the technical challenge of precise time synchronization between Android and iOS devices.

**Token Passing System (Deterministic Control)** [9]

The second alternative is a **Token System**, which shifts the network from a competitive model (where devices fight for the channel) to a cooperative model.

* **Mechanism:** A logical "Token" is passed between devices. At any given moment, exactly one device in the network holds the token.
* **Operational Logic:**
  + **Token Holder (Active State):** The device holding the token is the only node authorized to initiate major data transmission and reception sequences. It has exclusive rights to use the full bandwidth of the ultrasound channel to send its payload or handshake with a specific target.
  + **Non-Token Holders (Passive/Request State):** All other devices are prohibited from initiating data transfers. Their transmission capability is strictly limited to sending brief **"Token Request"** signals.
  + **Request Protocol:** If a non-token holder has data to send, it must wait in listening mode. It can only transmit a short request frame to the current token holder. Once the holder finishes its task, it passes the token to the requested device, granting it permission to transmit.
* **Advantage:** This method theoretically eliminates data packet collisions entirely, as no two devices can attempt to send full data frames simultaneously. It ensures an orderly flow of traffic, which is particularly useful in the high-interference environment of indoor ultrasound where "Hidden Node" problems are common.

**B. Development Framework Contingency:**

If our custom cross-platform implementation proves unstable, we will pivot to using established third-party libraries or frameworks that handle audio hardware abstraction. Instead of developing the low-level hardware access ourselves, we will integrate proven solutions to ensure reliability across Android and iOS.

**C. Signal Integrity Issues (General Failure):**

If communication remains unreliable due to environmental factors (and not the protocol or platform), we will implement Error Correction Codes (ECC) or reduce the data transmission rate (bitrate) to increase signal robustness against noise.

**1.5.** **Difficulties In Setting This Project**

**1.5.1.** **Algorithmic Challenges**

Collision Detection Constraints Implementing Standard Collision Detection (CSMA/CD), used in previous projects, was shown to be practically unfeasible in wireless acoustic networks due to hardware limitations. [9]

* **Self-Interference:** Mobile transducers operate in Half-Duplex mode. During transmission, the device’s own high-amplitude output saturates its receiver, making it impossible to detect incoming signals or interference simultaneously. [9]
* **Hidden Node Problem:** Two transmitters out of range of each other may transmit simultaneously to a central receiver, causing undetectable collisions at the destination. Conclusion: The system cannot rely on real-time detection and must instead implement Collision Avoidance (CSMA/CA) protocols like Request to send (RTS)/Clear To Send /(CTS). [3][6]

**Physical Layer Challenge:** Multipath Propagation & Inter-Symbol Interference (ISI)

* Indoor acoustic communication suffers from severe Multipath Propagation, where sound waves reflect off surfaces and reach the receiver at different times.
* **The Issue:** These delayed echoes overlap with subsequent data bits, causing Inter-Symbol Interference (ISI). This distorts the signal and can flip bits (0 vs 1).[4][5]

**Implementation Challenge:** Non-Deterministic OS Latency Developing for general-purpose mobile operating systems (Android & iOS) presents a Real-Time constraint.

* **The Issue:** Neither Android nor iOS are hard Real-Time Operating Systems (RTOS). System scheduling, background processes, and memory management can delay audio buffer delivery to the application layer. [9]

**The Risk:** These unpredictable delays cause Buffer Underflows, resulting in permanent loss of incoming signal segments

**1.5.1.1.** **How Do We Plan To Overcome Them?**

* Protocol (CSMA/CA): Since physical detection is impossible, we will implement an RTS/CTS handshake to logically reserve the medium before transmission. Additionally, Random Backoff timers will be used to manage retransmissions when the channel is busy.
* Signal Robustness: To mitigate Multipath interference, we selected a conservative symbol duration 0.5ms that allows echoes to decay. Data integrity is strictly enforced via 8-bit Checksum validation and an ACK confirmation.
* Multi-platform Management: To handle non-deterministic delays on Android and iOS, we will use multi-threaded architecture. This decouples the high priority recording task from processing, preventing data loss during OS lags.

**1.5.1.2.** **What Do We Do If Our Expectations Do Not Hold?**

* Fallback A: Lower Data Rate: If the Bit Error Rate (BER) is high due to echoes, we will increase the symbol duration (e.g., to 1.0ms), sacrificing speed for higher stability.
* Fallback B: Forward Error Correction (FEC): If retransmissions become too frequent, we will implement error-correcting codes (e.g., Hamming Code) to repair single-bit errors locally without resending packets.
* Fallback C: Protocol Simplification: If the RTS/CTS handshake creates too much latency for short messages, we will revert to a basic "Listen-Before-Talk" scheme without the handshake overhead.

**1.5.2.** **Hardware equipment**

**Mobile Devices (Nodes):** The system relies on standard smartphones running Android and iOS. The project must be tested on a variety of models (e.g., Samsung, Pixel, iPhone), as different manufacturers use different audio chipsets and microphone placements.

**Chapter 2.** **BACKGROUND AND RELATED WORK:**

**2.1 Reasons To Use Ultrasound**

There are many technologies that can help with short-range data transfer and communication such as Wi-Fi and Bluetooth and even Apple's AirDrop is based on these technologies. This technologies rely on RF which in some environments - may cause interferences - may not be efficient, suitable or applicable.  
 Ultrasound transfer may allow to overcome some of these limitations as high frequency soundwaves beyond the audible range of humans can be used to transfer small amounts of data over short distances while overcoming some of the limitations of RF transmissions without using specialized equipment such as radio transmitters.

For these reasons ultrasound transfer may offer an alternative to the current relevant technologies.

The limitations of RF data transfer that Ultrasound Data Transfer can overcome are Signal Interference and Spectrum Congestion limitations.  
Technologies like Wi-Fi and Bluetooth operate in specific, crowded frequency bands (e.g., 2.4 GHz, 5 GHz). [5]

This creates two major problems:

* **Signal Interference:**

RF Interference: Devices interfere with each other. A signal from several devices may collide with each other leading to dropped connections, slower speeds, and poor reliability.  
 Furthermore, RF signals are also susceptible to electromagnetic interference from common household and industrial electronics, which can disrupt communication.

### Ultrasound Overcomes this problem via not using RF waves at all as it simply uses sound waves instead, thereby unaffected by EM (Electromagnetic radiation), furthermore it avoids the crowded radio spectrum thereby doesn’t have to compete with a lot of other devices.

## Security and Signal Containment

**RF Security Risk:** RF signals are difficult to contain; they easily pass through solid objects such as walls, doors or windows. This is often beneficial for whole-home coverage, it also creates a significant security risk. A signal can be intercepted, or "snooped," by a third party outside the intended room or building, requiring complex encryption to secure the data.

**How Ultrasound Overcomes This:** Ultrasound waves are physical vibrations and are easily blocked by solid objects. This means the signal is physically confined to a single room. This property acts as a natural security feature, making it nearly impossible for someone in an adjacent room or outside the building to intercept communication[7], thus providing inherent privacy.

On that note, it's also important to mention that it's simple to determine the physical limits of this 'sound barrier.' The insecure area is strictly limited to the physical space where the soundwave can be detected, unlike RF signals, where it's difficult to know the boundary.

## 2.2 Specific Environmental Struggles:

### **RF Limitation**

RF waves propagate very poorly through conductive media, especially water. Saltwater heavily attenuates radio signals, making Wi-Fi and Bluetooth completely unusable for underwater communication.[11]

This same principle applies to the human body, which is composed of RF signals struggle to pass through tissue, requiring high power, which can be inefficient and potentially cause unsafe tissue heating.  
 Such issues make RF unusable for certain cases such as medical ones. [11]

### **How Ultrasound Overcomes It**

Ultrasound excels in the very environments where RF fails.

● Excellent Propagation in Liquids: As a mechanical wave, ultrasound travels exceptionally well through liquids. This is why it's the standard for SONAR (underwater) and medical imaging (in-body) [10].

● Ideal for In-Body Communication: It can be used for low-power, safe communication between medical implants and external devices, overcoming the high signal loss and heating concerns associated with RF [10].

**2.3 What Is Ultrasound Communication (Data Transmission Using Sound Waves Above The Human Hearing Range)?**

Ultrasound communication is a method of transmitting digital data using sound waves at frequencies higher than what humans can hear — typically above 20 kHz [11] (the upper limit of human hearing). Instead of using radio waves, ultrasound communication encodes information into high-frequency audio signals that can be played by a speaker and detected by a microphone.

How it compares to traditional wireless methods (Ultrasound Communication, Wi-Fi, Bluetooth, NFC (Near-Field Communication)).[5]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Technology | Use Case | Effective Range | Wave Type | Notes |
| Ultrasound Communication | Data transfer using high-frequency sound waves | ~2–5m | Acoustic (>20 kHz) | Works in liquids; blocked by walls (secure). |
| AirDrop | File sharing (Apple devices) | 10m | Bluetooth + Wi-Fi Direct | Uses Bluetooth for discovery, Wi-Fi for transfer. |
| Handoff / Continuity | Passing activity between Apple devices (Safari tabs, calls, clipboard) | 10m | Bluetooth + Wi-Fi Direct | Uses RF |
| Apple Pay (NFC) | Secure contactless payments | 13.56 MHz (RF) | Near-Field Communication (RF) | Extremely short-range radio. Designed for secure, authenticated exchanges. |

## 

## 

**Table 1: Comparison of Wireless Communication Technologies [5].**

## 2.4. Benefits Of Sending Data Over Sound: Summary

a.Operational Independence (Off-Grid Capability): as sound can operate on a peer-to-peer basis it does not require Sim card, any infrastructure and can work in areas where RF coverage does not exist.

b.Cost effectiveness: since we use sound to transfer data all that is required is a receiver, output device and the capability to compute the application, which means it can run on devices that do not have RF access (antenna).

c.Enhanced Security as seen in Section 2.1.2 since ultrasound waves have a short distance life, they offer privacy that RF cannot.

d.Environmental Resilience: as seen in Section 1.1.3  
ultrasound allows access through materials that therefore thereby be applied to transfer information where RF is lacking.

e.Ease of Integration: since the requirements for operations are so minimal the technology can be implemented on different platforms much more easily than other technologies.

**Chapter 3. GENERAL STEPS OF IMPLEMENTATION**

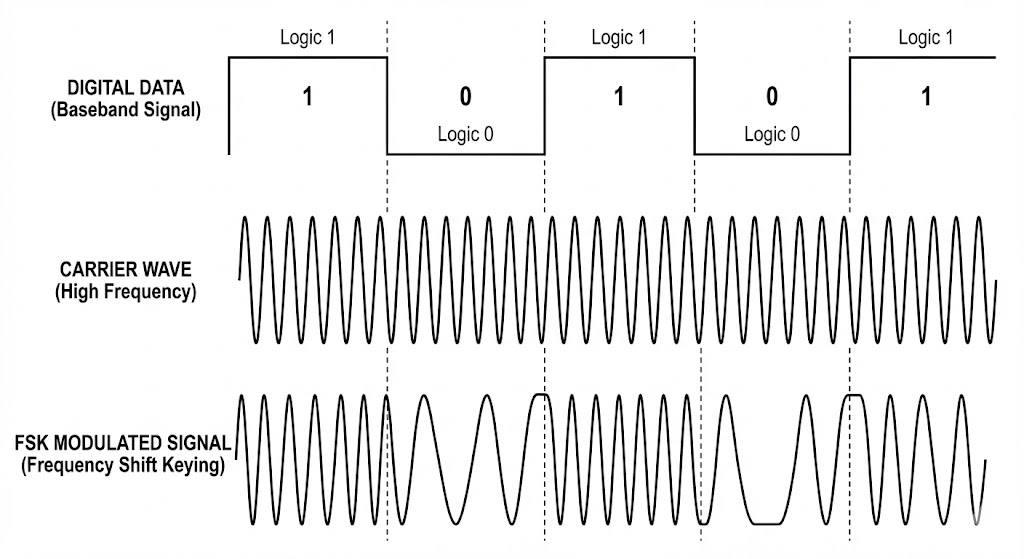
The implementation leverages the principle of Data-over-Sound, utilizing existing device hardware (microphone and speaker) to exchange data.

**3.1. Data Encoding (FSK):[1][2]**

The application uses Frequency-Shift Keying (FSK) to encode binary data onto ultrasonic sound waves.

Frequency Shift Keying (FSK) is a digital modulation technique extensively utilized in modern telecommunications and data transmission systems. It involves changing the frequency of a carrier signal to carry information.

The simplest FSK is binary FSK (BFSK, which is also commonly referred to as 2FSK or 2-FSK), in which the carrier is shifted between two discrete frequencies to transmit binary (0s and 1s) information.



**Figure 1:** Time-domain representation of Binary Frequency-Shift Keying (BFSK) modulation, illustrating the shift between carrier frequencies, (Figure Created by Google "nano banana pro" AI Model).

**3.1.1 Frequency Selection:**

The communication channel is set between 18 KHZ and 20 KHZ, which are generally inaudible to humans. A bit '0' is signified by transmitting on 18 KHZ, and a bit '1' is signified by transmitting on 20 KHz.

**3.1.2 Bit Duration:**

The optimal duration time for playing a tone representing one bit is 0.5 milliseconds. Piezoelectric components (used for ultrasonic transmission like speakers) behave like a spring, they take time to start vibrating at full amplitude and time to decay. At a typical frequency (~ 20kHz) with a single wave period is T = 50 microsec (T = 1/F). A standard speaker typically requires about 5 cycles to reach full amplitude (Rise Time). We can calculate the minimal Rise Time: 5 cycles \* 50 microsec = 0.25 milliseconds. To ensure the signal reaches a stable steady state after the initial rise time, and to allow for sufficient sampling by the receiver, a safety factor of 2 was applied 0.25 milliseconds \* 2 = 0.5 milliseconds.

**3.2 Connection Establishment:**

CSMA/CA with RTS/CTS:

CSMA is a basic method that controls the communication of multiple participants on a shared and decentralized transmission medium. However, this is now available in three different variants, which depend on the transmission medium. While CSMA/CA is mainly used in wireless networks, CSMA/CD was developed for Ethernet, and CSMA/CR is used in controller area networks (CAN), which are mainly used in cars and machines. [3]

Request to send and clear to send (RTS/CTS):

The frames “Request to Send” (RTS) and “Clear to Send” (CTS) are part of the optional extension CSMA/CA RTS/CTS. This procedure is upstream of the actual data transmission. If a participant determines that the transmission medium is free, the device first sends an RTS frame to the participant to receive the data. With this, the output computer makes it clear that it wants to start a transmission and will occupy the transmission medium for a certain time.

The receiver, in turn, sends a CTS frame to the original sender. As with the RTS frame, all other participants in the range are informed that the transmission is currently occupied and the transmitter is enabled for transmission. Only then does the original device start transmitting the data. Now it is not possible for the participants in a wireless network to detect collisions or other interference during transmission. For this reason, the receiving station needs to send an acknowledgement (ACK) when the data packet has arrived correctly.

If the ACK frame doesn’t appear, the sender of the data assumes that a complication has occurred and resends the data packet. The station has a preferential right to use the medium and doesn’t have to wait again for the channel to be free. The three frame types each consist of several fields.

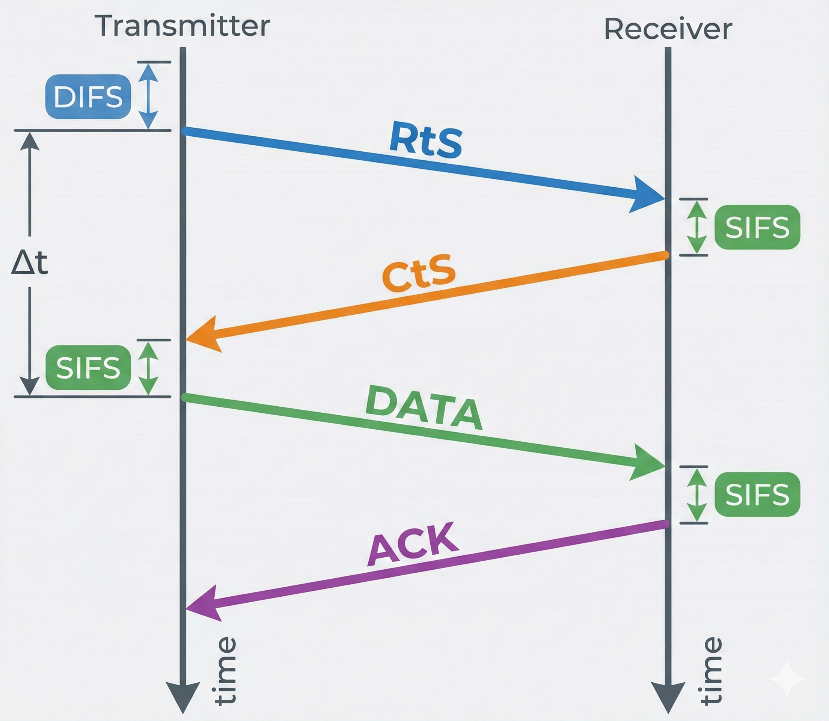
RTS/CTS Sequence:

**RTS (Request to Send):** The transmitter sends a short control frame indicating intent to transmit

**Wait (SIFS):** The transmitter enters a listening state for a predefined window (Timeout).

**CTS (Clear to Send):** If the receiver is idle, it responds with a CTS frame.

**Data Transmission:** Upon receiving the CTS, the transmitter sends the full data packet.



**Figure 2:** Sequence diagram of the CSMA/CA protocol utilizing the RTS/CTS handshake mechanism,(Figure Created by Google "nano banana pro" AI Model).

**3.2.1 RTS/CTS Frame Structure:**

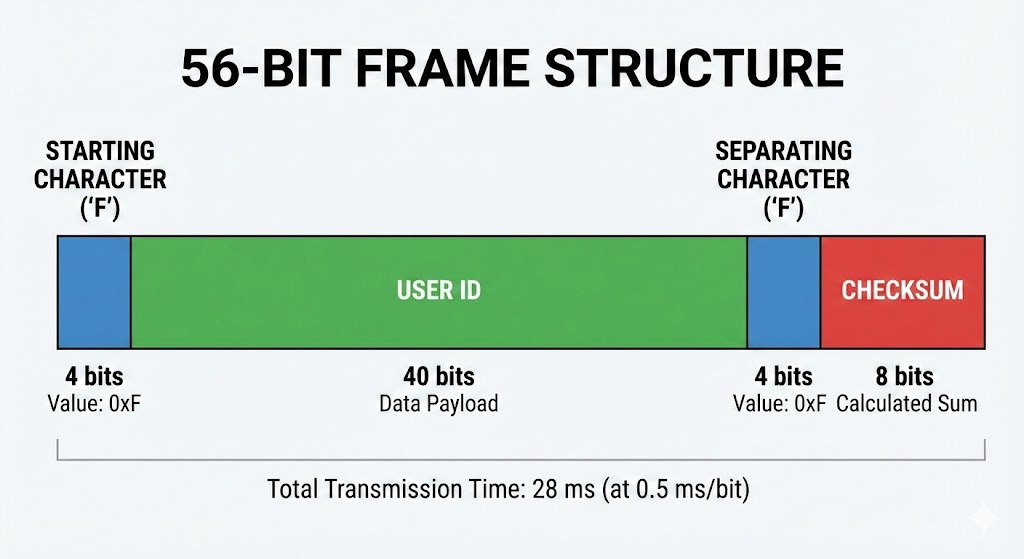
Each transmission is a fixed 56-bit frame. The transmission time for the full frame is approximately 28 milliseconds (56 \* 0.5ms). The frame components are:

1. Starting character 'F' (4 bits).

2. User ID (40 bits).

3. Separating character 'F' (4 bits).

4. Checksum (8 bits), calculated by summing the characters/digits of the data section.



**Figure 3:** Schematic breakdown of the 56-bit Ultrasonic RTS/CTS Frame, (Figure Created by Google "nano banana pro" AI Model).

**3.2.2 Transmitter Implementation:**

The transmission process uses the Android API's ‘AudioTrack’ or IOS API's ‘AVAudioEngine’.

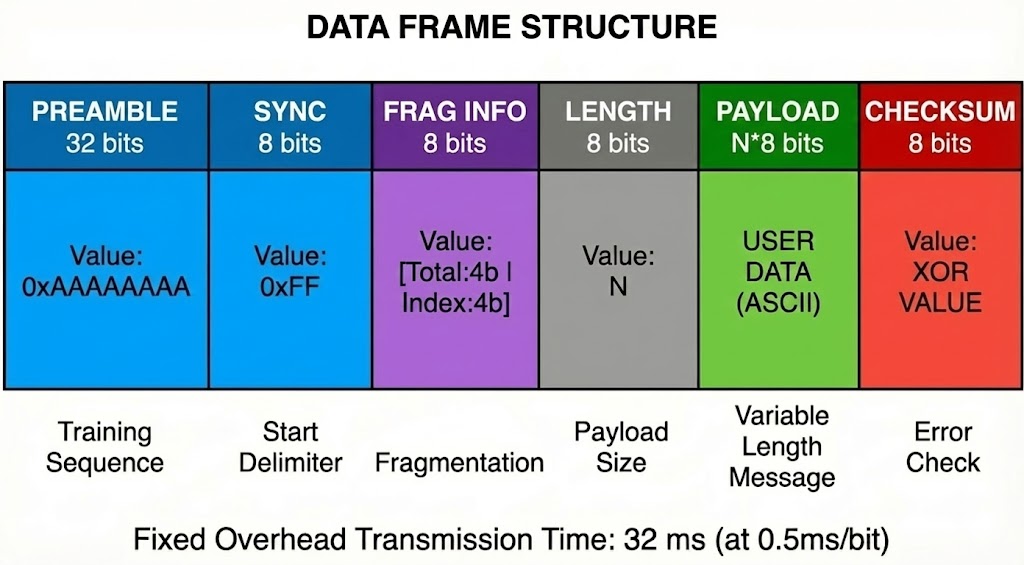
**Data Frame Structure:**

To accommodate the physical constraints of the ultrasonic channel (limited bandwidth and a transmission rate of 0.5ms per bit), the protocol utilizes a lightweight, byte-oriented frame structure. The design prioritizes minimal overhead while ensuring robust synchronization and support for multi-packet fragmentation.

The total fixed overhead per frame is **8 Bytes** (approx. 32ms transmission time), ensuring high throughput for user data.

| **Order** | **Field Name** | **Size** | **Value / Type** | **Function** |
| --- | --- | --- | --- | --- |
| **1** | **Preamble** | 4 Bytes | 0xAAAAAAAA | **Synchronization:** A training sequence enabling the receiver to perform clock recovery and stabilize AGC (Automatic Gain Control). |
| **2** | **Sync Word** | 1 Byte | 0xFF | **Start Delimiter:** A unique marker indicating the end of the training phase and the start of valid data. |
| **3** | **Frag Info** | 1 Byte | Bit-Packed | **Management:** Contains both the packet index and the total packet count. |
| **4** | **Length** | 1 Byte | 0x00 - 0x40 | **Payload Size:** Indicates the number of data bytes in the current frame (N). |
| **5** | **Payload** | N Bytes | ASCII Data | **User Data:** The actual text content being transmitted. |
| **6** | **Checksum** | 1 Byte | Calculated | **Integrity:** An XOR-based checksum for error detection. |

**Table 2: Frame Layout Table [6]**



**Figure 4:** Schematic breakdown of the Data Frame. (Figure Created by Google "nano banana pro" AI Model).

1. Tone Generation: The application iterates over the binary sequence (the frame) and assigns the corresponding high (20 KHZ) or low (18 KHZ) frequency tone.

2. Audio Output: The synthesized audio track is played with the highest amplitude to transmit the data.

3. Flow Control: Transmission is strictly controlled by the Collision Avoidance Protocol (Section 2) to ensure the channel is idle before sending.

Transmitted side pseudo code and algorithms:

**1.SYSTEM CONSTANTS & CONFIGURATION**

CONST SAMPLE\_RATE = 44100 // Hz

CONST BIT\_DURATION = 0.0005 // 0.5 ms

CONST SAMPLES\_PER\_BIT = 22 //

CONST FREQ\_LOGIC\_0 = 18000 // Hz

CONST FREQ\_LOGIC\_1 = 20000 // Hz

CONST RAMP\_PERCENT = 0.10 // 10% of bit duration for smoothing

// Protocol Constants

CONST PREAMBLE\_SEQ = 0xAAAAAAAA // 32-bit Training Sequence

CONST SYNC\_BYTE = 0xFF // Start Delimiter

**2. PACKET SERIALIZATION**

FUNCTION ConstructPacket(userMessageString, packetIndex, totalPackets):

// Initialize buffer

List<Byte> frameBuffer = new List()

// Add Preamble (4 Bytes)

frameBuffer.addInt32(PREAMBLE\_SEQ)

// Add Sync Word (1 Byte)

frameBuffer.addByte(SYNC\_BYTE)

// Add Fragmentation Info (1 Byte)

// Bit Packing: Upper Nibble = Total, Lower Nibble = Index

Byte fragInfo = (totalPackets << 4) | (packetIndex & 0x0F)

frameBuffer.addByte(fragInfo)

// Add Payload Length (1 Byte)

Byte length = userMessageString.length()

frameBuffer.addByte(length)

// Add Payload Data & Calculate Checksum

Byte checksum = 0

FOR each char IN userMessageString:

frameBuffer.addByte(char)

checksum = checksum XOR char

END FOR

// Add Checksum

frameBuffer.addByte(checksum)

// Serialization

// Convert the Byte List into a flat array of Bits

BitStream bits = ConvertBytesToBits(frameBuffer)

RETURN bits

END FUNCTION

**3. PHYSICAL MODULATION**

FUNCTION TransmitSignal(bitStream):

// Initialize Audio Hardware

AudioTrack track = InitAudioTrack(SAMPLE\_RATE, MODE\_STREAM)

track.play()

Double phase = 0.0 // Tracks phase to ensure continuity (FSK)

INT rampSamples = SAMPLES\_PER\_BIT \* RAMP\_PERCENT

// Transmission Loop

FOR each bit IN bitStream:

// Frequency Selection

IF (bit == 0) THEN

targetFreq = FREQ\_LOGIC\_0 // 18 kHz

ELSE

targetFreq = FREQ\_LOGIC\_1 // 20 kHz

END IF

// Sample Generation

Float buffer[SAMPLES\_PER\_BIT]

FOR j = 0 TO SAMPLES\_PER\_BIT - 1:

// Calculate angle based on previous phase

angle = 2 \* PI \* targetFreq \* (j / SAMPLE\_RATE)

sample = sin(angle + phase)

// Amplitude Ramping

IF (j < rampSamples):

sample = sample \* (j / rampSamples)

ELSE IF (j > SAMPLES\_PER\_BIT - rampSamples):

reversedIndex = SAMPLES\_PER\_BIT - j

sample = sample \* (reversedIndex / rampSamples)

END IF

buffer[j] = sample

END FOR

// Update Phase

phase = phase + (2 \* PI \* targetFreq \* BIT\_DURATION)

// Write to Hardware

track.write(buffer, SAMPLES\_PER\_BIT)

END FOR

// Cleanup

track.stop()

track.release()

END FUNCTION

**4. MAIN EXECUTION**

FUNCTION MainTransmit(textInput):

// Serialize Data

BitStream readyBits = ConstructPacket(textInput, 0, 1)

// Transmit Audio

TransmitSignal(readyBits)

END FUNCTION

**3.2.3. Receiver Implementation:**

The reception process uses the Android API's ‘AudioRecord’ or IOS API's ‘AVAudioRecorder’.

**1. Recording:** The device listens to and records audio from the input hardware.

**2. Decoding and Conversion:** The recorded audio data (in the time domain) is converted to the frequency domain using Fast Fourier Transform. This process identifies the dominant frequency in each time chunk, converting the analog signals back into binary digits ('0' or '1').

**3. Error Check:** Once the binary message is decoded, the receiver verifies the frame structure (start/separation flags) and the Checksum.

**4. Error Signal:** If the calculated checksum does not match the received checksum, or if flags are missing, the receiving device immediately transmits an error signal back to the sender.

**5. Local Storage:** If the checksum is verified, the data (other user's ID, time of encounter) is saved locally in the device's database. Encounters lasting 15 minutes or more are flagged for potential upload to the cloud.

**3.2.4 Pseudo Code And Algorithms:**

To demodulate the FSK signal, the system requires an efficient method to analyze frequency components within the received signal buffer. Two primary algorithms were evaluated:

**1. Fast Fourier Transform - FFT (Spectral Analysis):**

FFT provides a complete frequency spectrum analysis of the signal O(N\*log N).

**2. The Goertzel Algorithm (Targeted Detection):**

This algorithm is optimized for detecting specific, pre-known frequencies (in our case, 18kHz and 20kHz). It functions as a digital band-pass filter with extremely low computational complexity O(N).

Adaptive Strategy:

The primary implementation utilizes the FFT Algorithm due to its very well known, proven to be fast and reliable in the DSP world.

However, the system architecture retains the flexibility to switch to Goertzel, During the testing phase, empirical benchmarks will be conducted to verify that the 2kHz bandwidth separation (Guard Band) between Logical '0' (18kHz) and Logical '1' (20kHz) is sufficient for reliable detection over the acoustic channel.

**1. SYSTEM CONSTANTS & CONFIGURATION:**

CONST AudioSampleRate = 48000

CONST BitDurationSeconds = 0.0005 // 0.5 ms

CONST SamplesPerBit = 24 // 48000 \* 0.0005

CONST FFT\_WindowSize = 128

CONST NoiseThreshold = 500.0

// Calculated Frequency Indices for FFT [Index = (Freq \* FFT\_Size) / Rate]

CONST TargetIndex\_18k = 48 // (18000 \* 128) / 48000

CONST TargetIndex\_20k = 53 // (20000 \* 128) / 48000

// MAIN RECEIVER LOGIC

**2. FUNCTION StartReceiver():**

// Initialization

Instantiate AudioRecord as AudioInput

Float RawSampleBuffer[SamplesPerBit] // Size: 24

Complex ComplexFFTBuffer[FFT\_WindowSize] // Size: 128

String ReceivedBitStream = ""

Boolean IsListening = TRUE

// Start Hardware Capture

AudioInput.startRecording()

// Main Processing Loop

WHILE (IsListening):

// Read Raw Audio Chunk in Real-Time

SamplesReadCount = AudioInput.read(RawSampleBuffer, 0, SamplesPerBit)

// Zero Padding Prepare for FFT

// Extends the 24 samples to 128 to improve spectral resolution

FOR i = 0 TO FFT\_WindowSize - 1:

IF (i < SamplesPerBit):

ComplexFFTBuffer[i].Real = RawSampleBuffer[i]

ELSE:

ComplexFFTBuffer[i].Real = 0

END IF

ComplexFFTBuffer[i].Imaginary = 0

END FOR

// Converts time-domain samples to frequency domain

SpectrumOutput = FastFourierTransform(ComplexFFTBuffer)

// Get magnitude at specific target bins

Amplitude\_18k = Magnitude(SpectrumOutput[TargetIndex\_18k])

Amplitude\_20k = Magnitude(SpectrumOutput[TargetIndex\_20k])

// Demodulation Logic (Decision)

DetectedLogicBit = -1 // Default state (Idle)

// Check if signal is above noise floor

IF (Amplitude\_18k > NoiseThreshold OR Amplitude\_20k > NoiseThreshold):

// Compare which frequency is dominant

IF (Amplitude\_20k > Amplitude\_18k):

DetectedLogicBit = 1

ELSE:

DetectedLogicBit = 0

END IF

END IF

IF (DetectedLogicBit != -1):

ReceivedBitStream.append(DetectedLogicBit)

IF (PacketComplete(ReceivedBitStream)):

IsListening = FALSE

END IF

END IF

END WHILE

4. Cleanup

AudioInput.stop()

AudioInput.release()

END FUNCTION

**3.2.5 Collision Avoidance Algorithm (CSMA/CA):**

Since collision detection is nearly impossible in wireless acoustic networks, the system relies heavily on a specialized Carrier-Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol, designed to support a network of up to 15 devices in the optimal reception area (2-meter radius). The protocol utilizes unique, prime number-based waiting periods to minimize collision probability.

**A. Waiting Periods (IFG):**

Each device is assigned two unique prime numbers to calculate its waiting periods (Inter-Frame Gap, or IFG):

**1. Regular Base Waiting Period (RBWP):** A high prime number (range 2503–4493). This determines the main cycle period (approximately 5 minutes) between regular transmission attempts.

**2. Minimal Base Waiting Period (MBWP):** A lower prime number (range 59–1499). This is used as a shorter waiting period in case of a busy channel or a detected error/collision, to ensure the device gets another chance to transmit quickly (avoiding starvation).

**B. The CSMA/CA Protocol Flow:**

The device continually cycles between listening and transmitting based on the timer (RBWP or MBWP).

**1. Frame Ready (RBWP Timeout):** When the RBWP timer expires, the device prepares to transmit.

**2. Channel Check:** The device checks:if the frequency (18–20 KHZ) is idle (silent) or busy (not silent).

**3. If Busy:** The device immediately starts listenitofor MBWP. After MBWP, it checks the channel again. It remains in this MBWP loop until the channel is clear, preventing starvation.

**4. If Idle:** The device transmits the frame.

**5. Wait:** The sender enters listening to mode and waits for 2,000 milliseconds (2 seconds) for an error signal (collision/error detection) from any receiver.

**6. Collision Handling:** If an error signal is received: The sender concludes a collision or transfer error occurred. It switches to listening mode and sets the timer to the shorter MBWP before attempting a re-transmission.

If no error signal is received: The transmission was successful. The device resets the timer to RBWP for the start of the next 5-minute cycle.

**3.2.6 Cross-Platform Implementation (Android vs. iOS):**

**A. Android Implementation:**

The entire project is explicitly an Android Based Application. The implementation relies on the Java\Kotlin libraries and the Android Integrated Development Environment (IDE).

Key Components: The core functions utilize Android-specific APIs (`AudioTrack` for playing sound and `AudioRecord` for recording sound). The application is designed to run as a Service in the background (24/7).

**B. iOS Implementation:**

**Low-Level AudioRecord/AudioTrack Equivalent:** Core Audio (Remote I/O Unit): The most direct, low-latency access to audio hardware.

**High-Level Equivalent:**

**AVFoundation (AVAudioEngine):** A modern, simpler API for managing input (inputNode) and output (outputNode).

**Key Challenge (24/7 Background):** iOS heavily restricts background execution.

The potential solution is the audio Background Mode, but this is subject to system limitations and strict Apple review.

**Chapter 4**. EXPECTED RESULTS:

The primary objective of this project is to design and implement a fully functional acoustic communication network, utilizing ultrasonic waves as the physical transmission medium.

The system is created as a cross-platform mobile application (compatible with both Android and iOS), enabling seamless communication between heterogeneous devices.

Key Technical Specifications:

Protocol: The Data Link Layer implements a collision avoidance mechanism based on the RTS/CTS (Request to Send / Clear to Send) handshake protocol to ensure reliable data delivery.

Capacity & Range: The network is designed to support a cluster of at least 3 concurrent nodes within an optimal effective radius of 2 meters.

Topology: The system supports a many-to-many communication architecture, managing data transfer from multiple emitters to multiple recipients simultaneously.

**Chapter 5**. UML

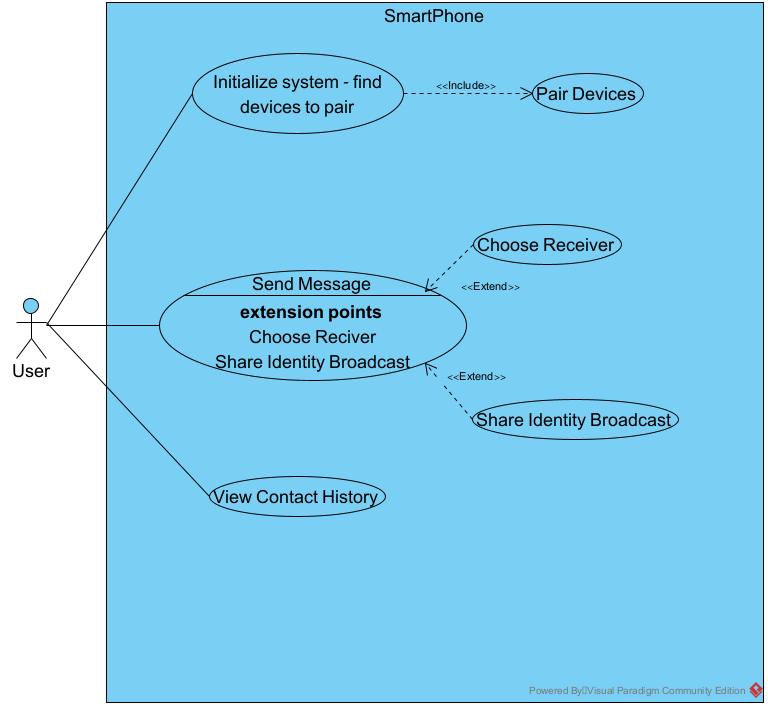


Figure 4: Use Case: Many-to-Many Interaction and Messaging Protocol

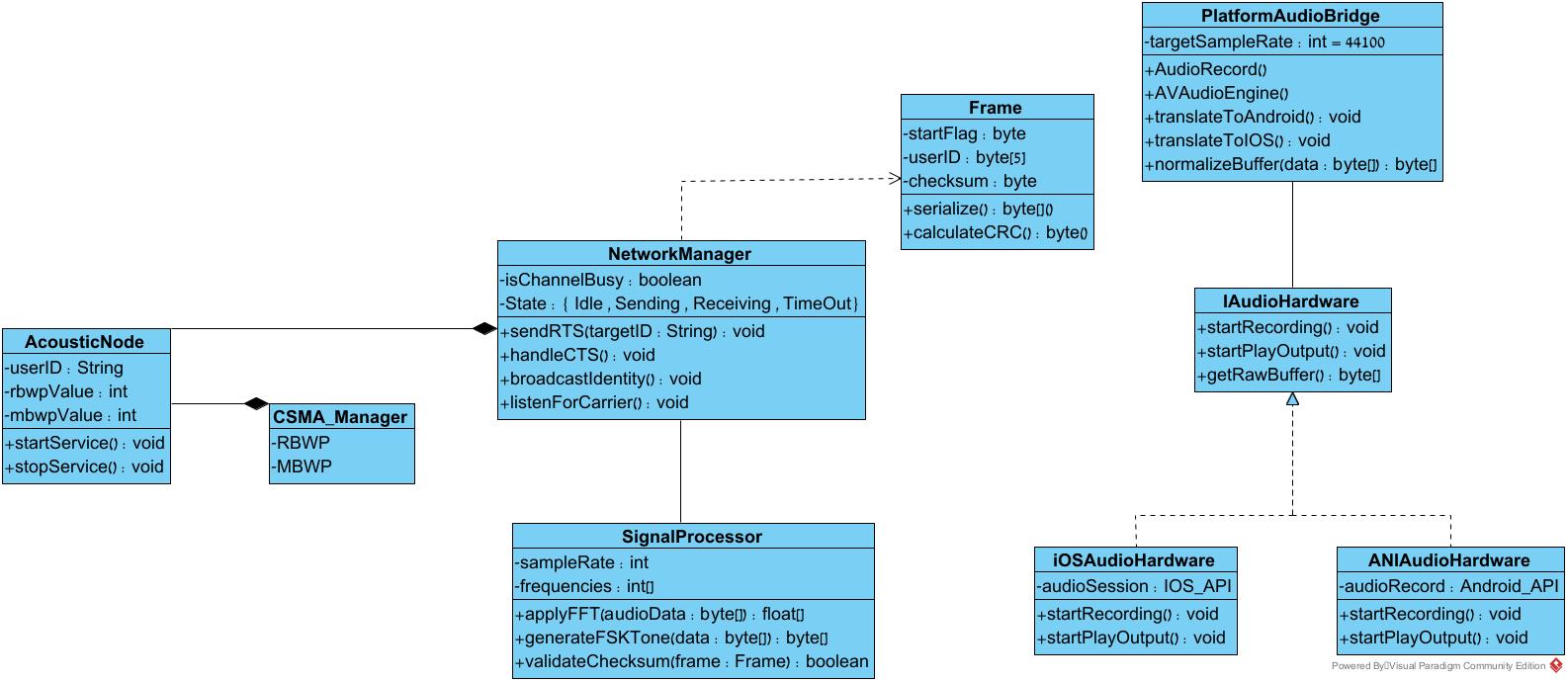


Figure 5: Class Diagram: Structural Design of the CSMA/CA and Signal Processing Layers

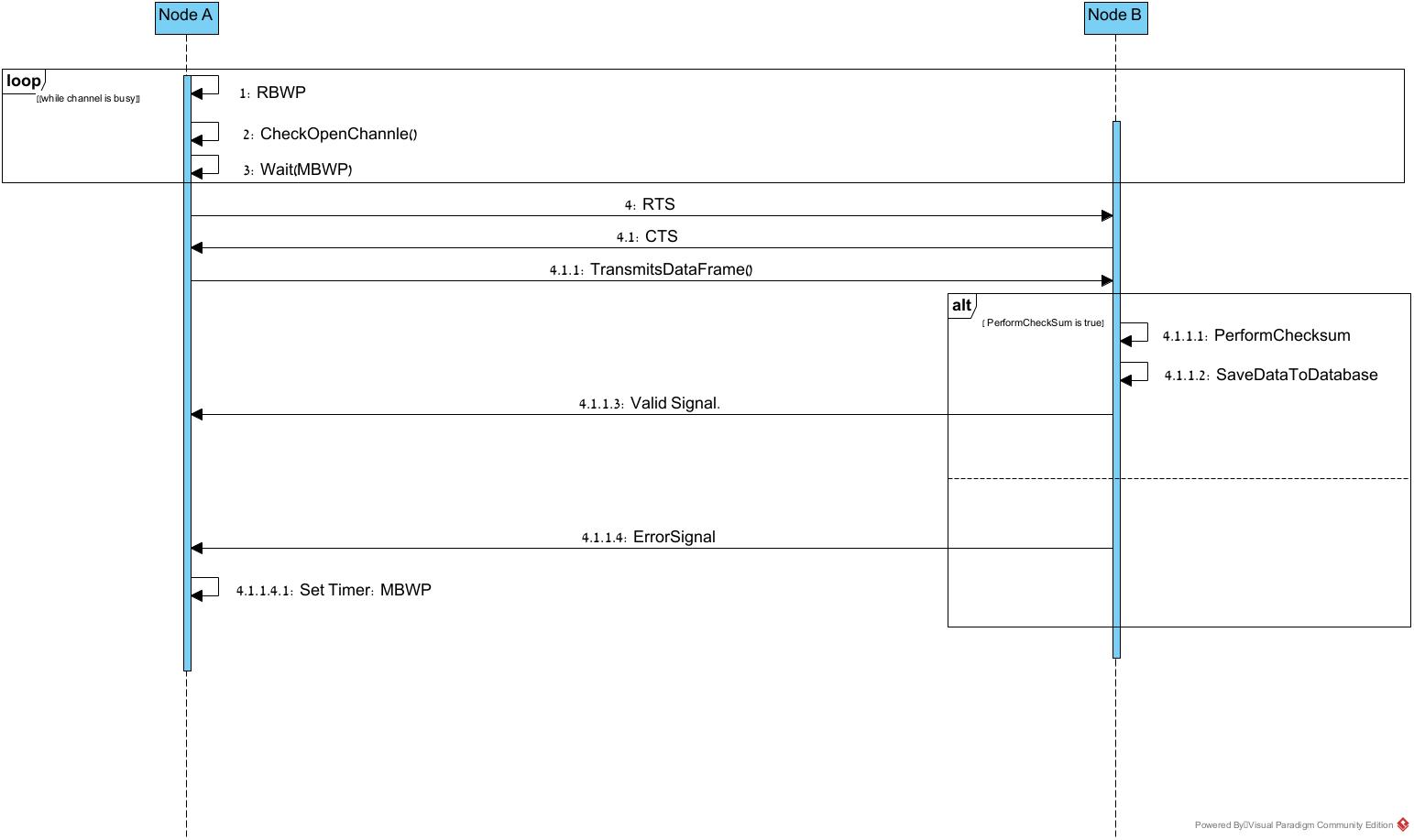


Figure 6: Sequence Diagram: CSMA/CA Protocol with RTS/CTS Handshake Sequence

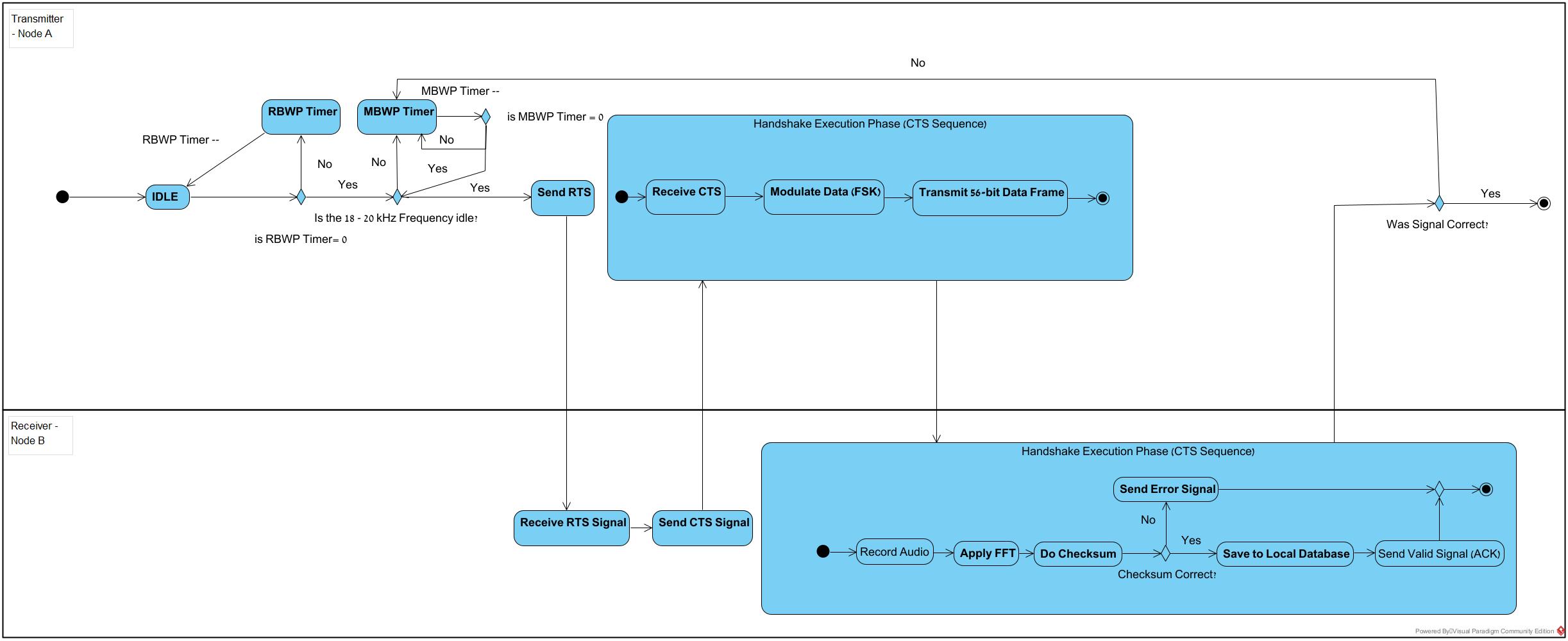


Figure 7: Activity Diagram: CSMA/CA Protocol with RTS/CTS Handshake

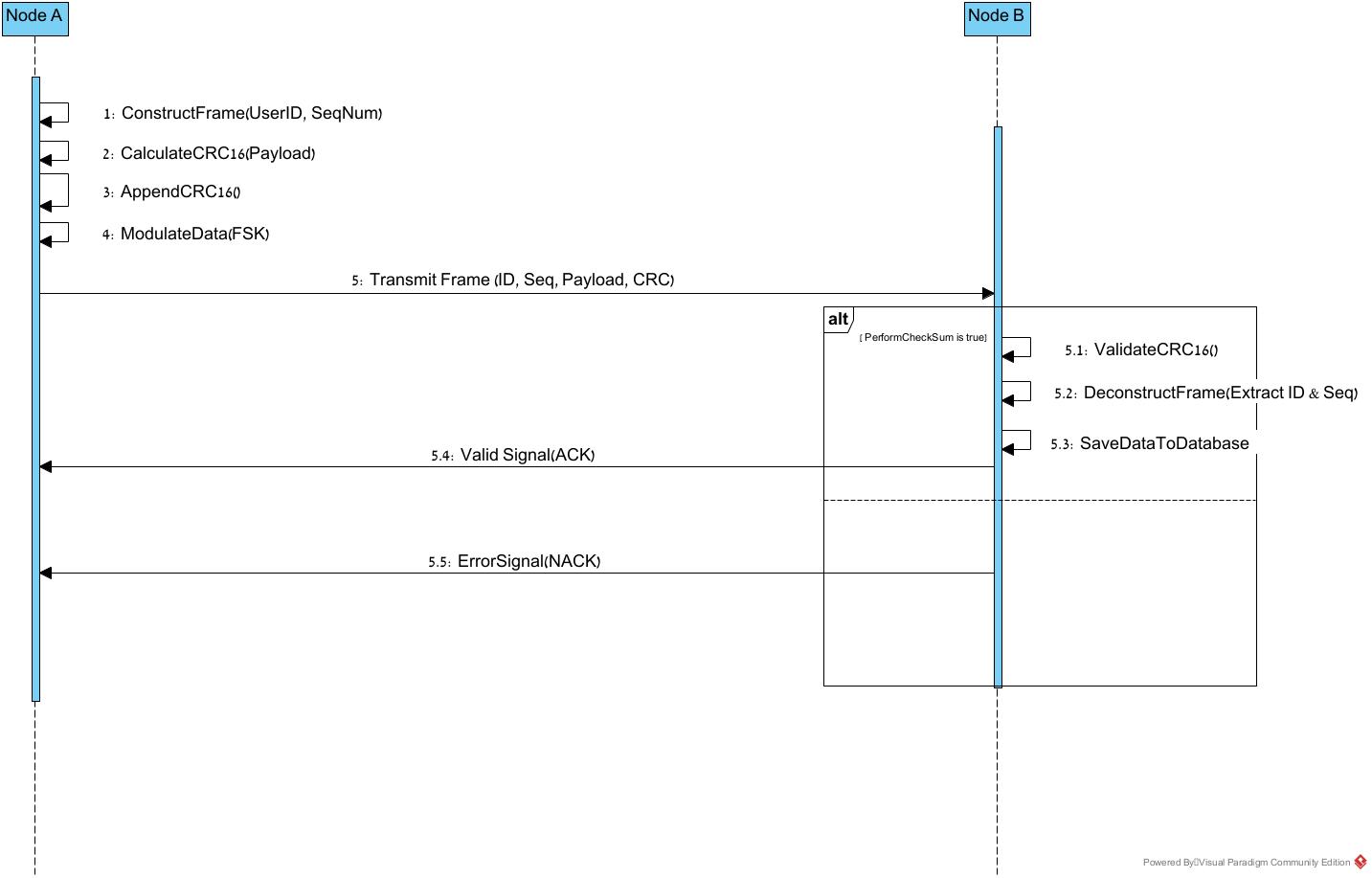


Figure 8: Sequence Diagram: Data Transmission and Frame Verification Protocol

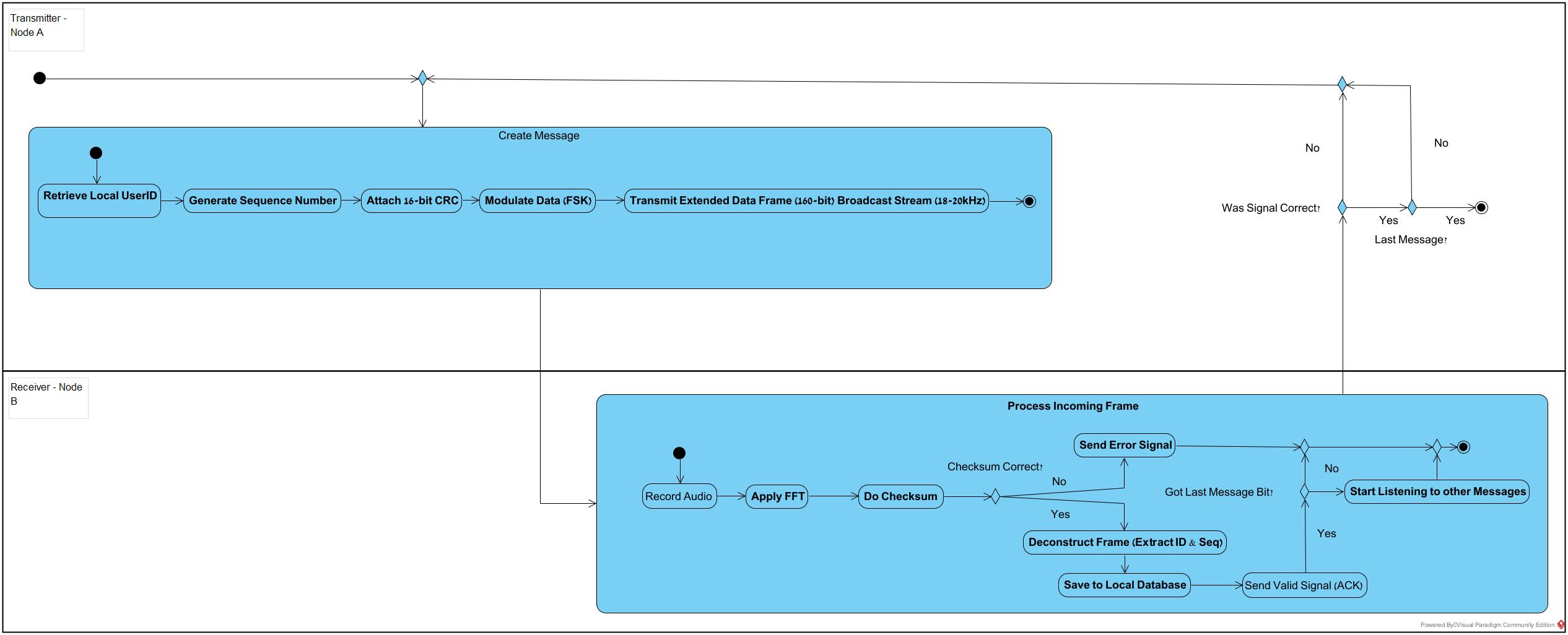


Figure 9: Activity Diagram: Data Framing and Physical Layer

**Chapter 6**. Testing:

**6.1. Bit Packing And Fragmentation Logic**

* **Objective:** Verify that the ConstructPacket function correctly packs the "Total Packets" and "Current Index" integers into a single byte.
* **Procedure:** Input Total=3, Index=2. Run the bit-packing logic.
* **Success Criteria:** The resulting byte must equal 0x32 (Binary 00110010).

**6.1.1. Checksum Calculation Accuracy**

* **Objective:** Validate the integrity of the XOR checksum algorithm.
* **Procedure:** Manually calculate the XOR sum of the string "TEST". Run the code’s checksum function on the same string.
* **Success Criteria:** The code output must match the manual calculation exactly.

**6.1.2 Packet Serialization**

* **Objective:** Ensure that the raw string is correctly converted into the binary protocol frame.
* **Procedure:** Pass the string "TEST" to the serializer. Inspect the output bit array.
* **Success Criteria:** The array must follow the strict order: Preamble -> Sync -> Frag -> Length -> Payload -> Checksum.

**6.1.3 Frequency Analysis Algorithm**

* **Objective:** Test the FFT logic using a mathematically generated array
* **Procedure:** Feed an array containing a pure mathematical sine wave of 18kHz into the detection function.
* **Success Criteria:** The algorithm must return a significantly higher magnitude for the 18kHz bin compared to the 20kHz bin.

**6.2 Physical Layer Testing**

**6.2.1. Bit Duration And Timing**

* **Objective:** Confirm that each bit lasts exactly 0.5ms as defined.
* **Procedure:** Record the transmission of a single bit using an external PC with audio editing software (Audacity). Measure the waveform length.
* **Success Criteria:** The waveform duration must be 0.5ms

**6.3. Integration Testing**

**6.3.1. Single Packet Transmission**

* **Objective:** Test the successful transfer of a short message.
* **Procedure:** Tx sends the string "TEST".
* **Success Criteria:** Rx decodes the message correctly, checksum validation passes, and the text "TEST" appears on the screen.

**6.3.2. RTS/CTS Handshake Verification**

* **Objective:** Ensure the collision avoidance mechanism works.
* **Procedure:** Tx initiates a transmission.
* **Success Criteria:** Tx waits for CTS. Rx sends CTS upon hearing RTS. Tx only begins data transmission *after* receiving the CTS.

**6.4. Field Testing**

**6.4.1 Maximum Effective Range**

* **Objective:** Determine the maximum distance for reliable communication.
* **Procedure:** Place devices 10cm apart and increase distance in 10cm increments until decoding fails.
* **Success Criteria:** Document the distance where communication is failed.

**6.4.2 Noise Immunity**

* **Objective:** Validate performance in a noisy environment.
* **Procedure:** Perform transmission while sending external noise near the receiver.
* **Success Criteria:** The system should successfully filter out the noise

**6.4.3 Wall/Barrier Penetration Test**

* Objective: Determine the signal's ability to diffract around or penetrate obstacles, verifying if direct Line-of-Sight (LOS) is mandatory.
* Procedure:
  1. Establish a stable connection between devices at a distance of 2 meters.
  2. Place obstacles of increasing density directly between the speaker and microphone:
     + Level 1: A sheet of paper.
     + Level 2: A human body.
     + Level 3: A solid wooden door or wall.
* Success Criteria:
  1. Pass: The system must accurately report signal loss or checksum failure when the signal is blocked. It must not crash or display random garbage data.

**6.4.4. Battery And Thermal Stability**

* **Objective:** Ensure the app does not crash or overheat during continuous use.
* **Procedure:** Run a continuous transmission loop for 10 minutes.
* **Success Criteria:** App remains stable, no "Application Not Responding" errors.

Chapter 7. FR\NFR REQUIREMENTS:

Functional Requirements

1. The system must utilize Binary Frequency-Shift Keying (BFSK) to encode digital data into ultrasonic tones between 18 kHz and 20 kHz.
2. The system must capture real-time audio and apply Fast Fourier Transform (FFT) or the Goertzel algorithm to accurately decode binary bits.
3. The system must implement a "Listen-Before-Talk" mechanism utilizing unique, prime-number-based backoff timers (RBWP and MBWP) to prevent data collisions.
4. The system must perform a Request to Send (RTS) and Clear to Send (CTS) sequence to reserve the channel and eliminate the "Hidden Node Problem”.
5. The system must append a 16-bit CRC checksum to each 160-bit frame, and the receiver must validate this checksum before processing the data.
6. The system must return a "Valid Signal" (ACK) upon successful validation or an "Error Signal" (NACK) to trigger retransmission.
7. The system must support a many-to-many architecture capable of managing at least 3 concurrent nodes within a 2-meter radius.
8. The system must validate user encounters (ID and timestamp) must be committed to a local device database.

Non-Functional Requirements

1. Transmission must remain above the standard human hearing threshold (typically 18 kHz+) to ensure a non-intrusive user experience.
2. Each bit must have a duration of 0.5ms to allow for signal rise time and to mitigate Inter-Symbol Interference (ISI) caused by echoes.
3. The system must achieve successful data transfer in the presence of ambient noise and multipath propagation.
4. The network must operate on a peer-to-peer basis without requiring SIM cards, Wi-Fi infrastructure, or cellular coverage.
5. The system must operate at safe power levels for the human body, avoiding the tissue-heating concerns associated with high-power RF.

Chapter 8. Reference

[1] ref: <https://www.rfpage.com/what-is-frequency-shift-keying-fsk/>

[2] ref: <https://en.wikipedia.org/wiki/Frequency-shift_keying>

[3] ref: <https://www.ionos.com/digitalguide/server/know-how/csmaca-carrier-sense-multiple-access-with-collision-avoidance/>

[4] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York, NY, USA: McGraw-Hill, 2008.

[5] ref: RF Wireless World, "NFC vs. RFID vs. Bluetooth vs. Wi-Fi: Key Differences Explained," *rfwireless-world.com*. [Online]. Available: <https://www.rfwireless-world.com/terminology/nfc-vs-rfid-vs-bluetooth-vs-wifi>.

[6] **Miras Safadi** and **Rani Hassan**,  **Smart Visit Card (SVC)**

**An Android-Based Application for Managing Visit Cards**

” Final Project Report, Dept. Software Eng., Braude College of Engineering, Karmiel, Israel.

[7] **Tal Zilberman** and **Ariela Havkin**,” **Ultra Sound Data Transfer Many To Many”**

," Final Project Report, Dept. Software Eng., Braude College of Engineering, Karmiel, Israel.

[8] **Taimor Fares** and **Fares Jaraisy,** **Smart Visit Card**

Inaudible Information Transfer between Android Based Devices

," Final Project Report, Dept. Software Eng., Braude College of Engineering, Karmiel, Israel.

[9] Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5th ed.). Pearson.

[10] Szabo, T. L. (2004). *Diagnostic Ultrasound Imaging: Inside Out*. Elsevier Academic Press. [11]https://www.researchgate.net/publication/304130724\_High\_Data\_Rate\_Ultrasonic\_Communications\_for\_Wireless\_Intra-body\_Networks