UART Communication Stress test on NodeMCU ESP8266

Name: Halima Sadia
Student ID: 2021-3-60-280
Course Name: CSE406
Date: July 25, 2025

1. Abstract

This report evaluates UART communication performance between two NodeMCU ESP8266 boards, quantifying throughput, message speed, and error rates. Tests varied baud rates (9600, 38400, 115200 bps), message sizes (10, 50, 100 bytes), and
intervals (0, 10, 100 ms). SoftwareSerial proved robust at 9600 and 38400 bps (0.00% errors) but critically unreliable at 115200 bps due to timing limitations. The optimal SoftwareSerial configuration was 38400 baud with 100-byte messages at 0ms interval, achieving 1919.00 bytes/second throughput with perfect data integrity. The study emphasizes hardware UART for high-speed, reliable serial communication on ESP8266.

2. Introduction

UART is a fundamental serial protocol crucial for embedded systems. This lab aimed to stress test UART communication between two NodeMCU ESP8266 boards, measuring throughput, transfer speed, and error rate. We systematically varied baud rates (9600, 38400, 115200), message sizes (10, 50, 100 bytes), and transmission intervals (0ms, 10ms, 100ms) to assess performance and identify optimal settings.

3. Experimental Design & Setup

Hardware Configuration:
Two NodeMCU ESP8266 boards (Master and Slave) were connected via expansion boards and a breadboard. SoftwareSerial pins (Master D5 TX to Slave D6 RX; Master D6 RX to Slave D5 TX) were used, with a shared GND. Initial wiring challenges were resolved through careful verification.
Software Implementation:
Arduino IDE was used to program both NodeMCUs. The Master (NodeMCU1_Master_StressTest.ino) sent sequenced messages, verified echoes, and calculated metrics. The Slave (NodeMCU2_Slave_StressTest.ino) echoed received messages and dynamically adjusted its SoftwareSerial baud rate upon receiving "BAUD: [baudrate]" commands from the Master. All communications and results were logged to USB Serial Monitors. The trim() function was used to ensure accurate string comparisons.

4. Performance Metrics and Observations

The stress test involved 27 scenarios, each 10 seconds long. Results are summarized below:

Observed Performance Metrics:

Baud Rate	Message Size (bytes)	Interval (ms)	Messages Sent	Messages Received	Errors	Error Rate (%)	Throughput (bytes /second)	Message Rate (messages/second)
9600	10	0	489	489	0	0.00	478.00	48.90
9600	10	10	331	331	0	0.00	320.00	33.10
9600	10	100	85	85	0	0.00	75.50	8.50
9600	50	0	99	99	0	0.00	484.10	9.90
9600	50	10	90	90	0	0.00	440.00	9.00
9600	50	100	50	50	0	0.00	244.00	5.00
9600	100	0	49	49	0	0.00	484.10	4.90
9600	100	10	47	47	0	0.00	464.30	4.70
9600	100	100	33	33	0	0.00	325.70	3.30
38400	10	0	1804	1804	0	0.00	1873.40	180.40
38400	10	10	655	655	0	0.00	644.00	65.50
38400	10	100	96	96	0	0.00	85.40	9.60
38400	50	0	384	384	0	0.00	1909.00	38.40

38400	50	10	278	278	0	0.00	1379.00	27.80
38400	50	100	80	80	0	0.00	391.00	8.00
38400	100	0	193	193	0	0.00	1919.00	19.30
38400	100	10	162	162	0	0.00	1609.00	16.20
38400	100	100	66	66	0	0.00	652.40	6.60

Note: 115200 baud showed severe instability, leading to frequent timeouts and corrupted messages, rendering quantitative metrics unreliable for SoftwareSerial.

Qualitative Observations:
Both 9600 and 38400 baud consistently yielded 0.00% error rates, indicating robust communication. In contrast, 115200 baud exhibited a very high error rate, with frequent "Mismatch error" (truncated/corrupted messages like \X) and "Timeout error" messages, making it unsuitable for reliable data transfer via SoftwareSerial. Throughput and message rate increased as intervals decreased. Larger messages generally increased throughput but decreased message rate. Higher baud rates (e.g., 38400 vs. 9600) significantly boosted both metrics when reliable.

5. Discussion & Interpretation

Performance Dynamics: Throughput correlates directly with baud rate and message size; higher values increase data transfer efficiency. Message rate is maximized with smaller messages and shorter intervals. This highlights the trade-off between maximizing data volume (throughput) and the number of individual transactions (message rate).

Reliability Assessment: SoftwareSerial is highly reliable at 9600 and 38400 baud (0.00% errors). However, at 115200 baud, SoftwareSerial fails due to the ESP8266's CPU struggling with precise timing amidst other background tasks (e.g., Wi-Fi stack). This leads to data corruption (e.g., \X characters) and buffer overflows, causing "Mismatch error" and "Timeout error." The trim() function was vital for accurate string comparisons.

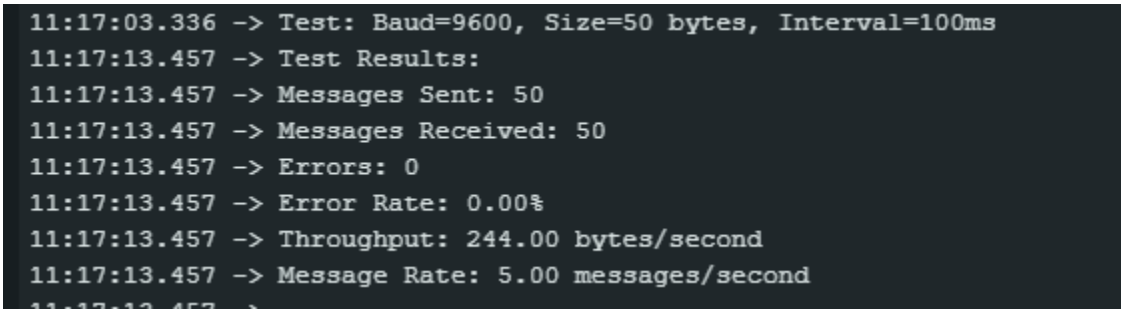
Optimal Configuration: The optimal SoftwareSerial configuration for this setup is

38400 baud with **100-byte messages** and a **0ms interval**, achieving **1919.00 bytes/second throughput** at **0.00% error rate**.

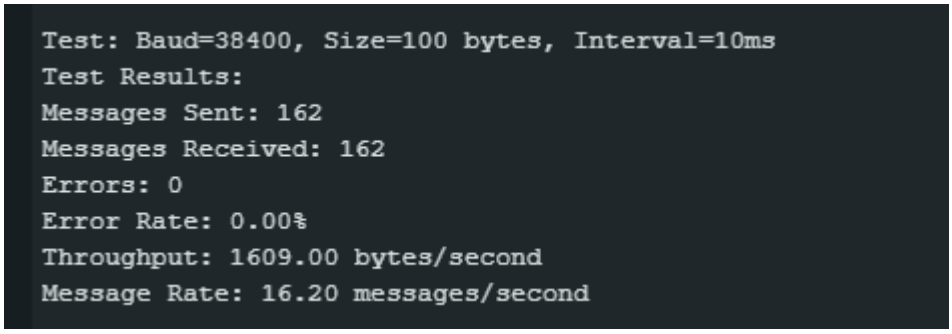
Challenges: Initial challenges included correct SoftwareSerial wiring and resolving a baud rate synchronization issue where the Slave incorrectly read commands from USB Serial instead of SoftwareSerial. The most persistent challenge was the inherent unreliability of SoftwareSerial at 115200 baud.

Figures:

- **Figure 1: Master NodeMCU Serial Monitor Output (Example – 9600 Baud, 50 bytes, 100ms interval)**



- **Figure 2: Master NodeMCU Serial Monitor Output (Example – 38400 Baud , 100 bytes, 10ms interval)**



6. Conclusion

This lab successfully demonstrated and quantified the performance of UART communication between two NodeMCU ESP8266 boards under various conditions. Key findings indicate that SoftwareSerial provides highly reliable (0.00% error rate) communication at baud rates of 9600 and 38400. At 38400 baud, a 100-byte message size with a 0ms interval yielded the highest throughput of 1919.00 bytes/second, showcasing the maximum practical data transfer rate for this setup without errors. However, the limitations of SoftwareSerial became critically apparent at 115200 baud, where communication suffered from severe data corruption and timeouts, rendering it unsuitable for reliable data transfer. This highlights that while

UART is a robust protocol for data transfer, the choice between hardware and software implementations significantly impacts performance and reliability at higher speeds. For future improvements, utilizing the ESP8266's dedicated hardware UART (UART0 or UART1) is strongly recommended for applications requiring 115200 baud or higher. Additionally, improving wiring quality and considering error correction protocols could further enhance robustness in noisy environments.