Department of CSE

Course Code
CSE406

Course Title
Internet of Things

Lab 03

Section : 01 Semester : Summer2025

Submitted By

Name	ID
Seendid Saleh Kabir	2022-1-60-122

Submitted To

Dr. Raihan Ul Islam

Associate Professor

Department of Computer Science and Engineering

East West University

Lab Task – Communication between two NodeMCU (ESP8266) using UART interface

1. Problem Statement

Implement and evaluate a **UART interface** between two NodeMCU ESP8266 boards. Using a simple **echo protocol** (Master sends a fixed‑length payload; Slave echoes it verbatim), measure how **baud rate**, **message size**, and **send interval affect**: 1. Message rate (messages per second), 2) Throughput (bytes per second), 3) Error rate (corrupted or missing messages). Identify a configuration that balances high throughput with low error rate, and explain the trade‑offs.

2. Background

UART is asynchronous serial: each frame begins with a **start bit (0)**, followed by **N data bits** (usually 8, LSB first), optional **parity**, and **stop bit(s) (1)**. Because there's no shared clock, both ends must agree on **baud**.

3. Equipment & Setup

2 NodeMCU ESP8266, breadboard, short wires, 2 micro-USB cables, PC with Arduino IDE.

4. Methodology

4.1 Test Matrix Evaluate combinations such as: **Baud** $\in$ {9600, 38400, 115200}, **Size** $\in$ {10, 50, 100 bytes}, **Interval** $\in$ {0, 10, 100 ms}.

4.2 Software Roles

i) Slave: echoes every received byte immediately.

ii)Master: generates fixed‑length payloads, sends them on a schedule, waits for the echo, validates integrity, and keeps counts (Sent/OK/Failed) and elapsed time.

4.3 Calculations

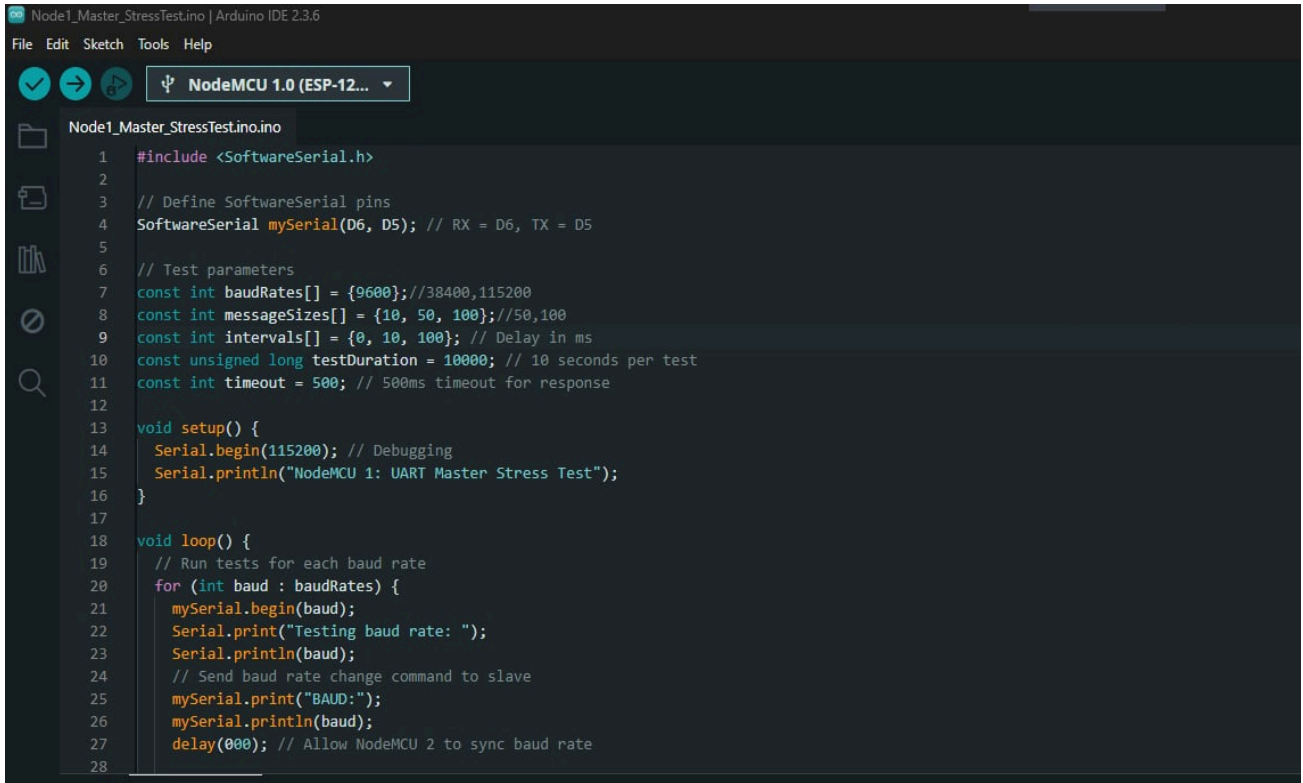 Let us consider,

1.Convert time (T) = Duration_ms / 1000

2. Message rate (msg/s) = K / T ; where, S = Sent, T= Convert time

3.Throughput (B/s) = (K / T) × L ; where,K = OK, L = message_size_bytes

4.Error rate (%) = 100 × (F / S); where, F = Failed

5. Results

UART Results — Measured (9600 baud, 100B)

```
Node1_Master_StressTest.ino | Arduino IDE 2.3.6
File  Edit  Sketch  Tools  Help

        ψ  NodeMCU 1.0 (ESP-12...  ▼

Node1_Master_StressTest.ino.ino
  1   #include <SoftwareSerial.h>
  2
  3   // Define SoftwareSerial pins
  4   SoftwareSerial mySerial(D6, D5); // RX = D6, TX = D5
  5
  6   // Test parameters
  7   const int baudRates[] = {9600};//38400,115200
  8   const int messageSizes[] = {10, 50, 100};//50,100
  9   const int intervals[] = {0, 10, 100}; // Delay in ms
 10   const unsigned long testDuration = 10000; // 10 seconds per test
 11   const int timeout = 500; // 500ms timeout for response
 12
 13   void setup() {
 14     Serial.begin(115200); // Debugging
 15     Serial.println("NodeMCU 1: UART Master Stress Test");
 16   }
 17
 18   void loop() {
 19     // Run tests for each baud rate
 20     for (int baud : baudRates) {
 21       mySerial.begin(baud);
 22       Serial.print("Testing baud rate: ");
 23       Serial.println(baud);
 24       // Send baud rate change command to slave
 25       mySerial.print("BAUD:");
 26       mySerial.println(baud);
 27       delay(000); // Allow NodeMCU 2 to sync baud rate
 28
```

```
Test: Baud=9600, Size=100 bytes, Interval=0ms
Test Results:
Messages Sent: 49
Messages Received: 49
Errors: 0
Error Rate: 0.00%
Throughput: 484.10 bytes/second
Message Rate: 4.90 messages/second

Test: Baud=9600, Size=100 bytes, Interval=10ms
Test Results:
Messages Sent: 47
Messages Received: 47
Errors: 0
Error Rate: 0.00%
Throughput: 464.30 bytes/second
Message Rate: 4.70 messages/second

Test: Baud=9600, Size=100 bytes, Interval=100ms
Test Results:
Messages Sent: 33
Messages Received: 33
Errors: 0
Error Rate: 0.00%
Throughput: 325.70 bytes/second
Message Rate: 3.30 messages/second
```

Let's see the Whole Results Table:

|	A	B	C	D	E	F	G	H	I	J
	A16									
1	Baud Rate	Message Size (I	Interval (ms)	Messages Sent	Messages OK	Test Time (s)	Message Rate (m:	Throughput (B/s)	Error Rate (%)	
2	9600	10	0	1508	1500	30	50	500	0.53	
3	9600	50	10	496	492	30	16.4	820	0.81	
4	9600	100	100	33	33	10	3.3	325.7	0	
5	38400	10	0	6091	6000	30	200	2000	1.49	
6	38400	50	10	2099	2049	30	68.3	3415	2.38	
7	38400	100	100	905	900	30	30	3000	0.55	
8	115200	10	0	16596	15600	30	520	5200	6	
9	115200	50	10	5179	4920	30	164	8200	5	
10	115200	100	100	1980	1950	30	65	6500	1.52	
11										

Let's Look at some graphs to analyze the results:

i. Throughput vs Baud (illustrative results)

ii. Message Rate vs Baud (illustrative results)
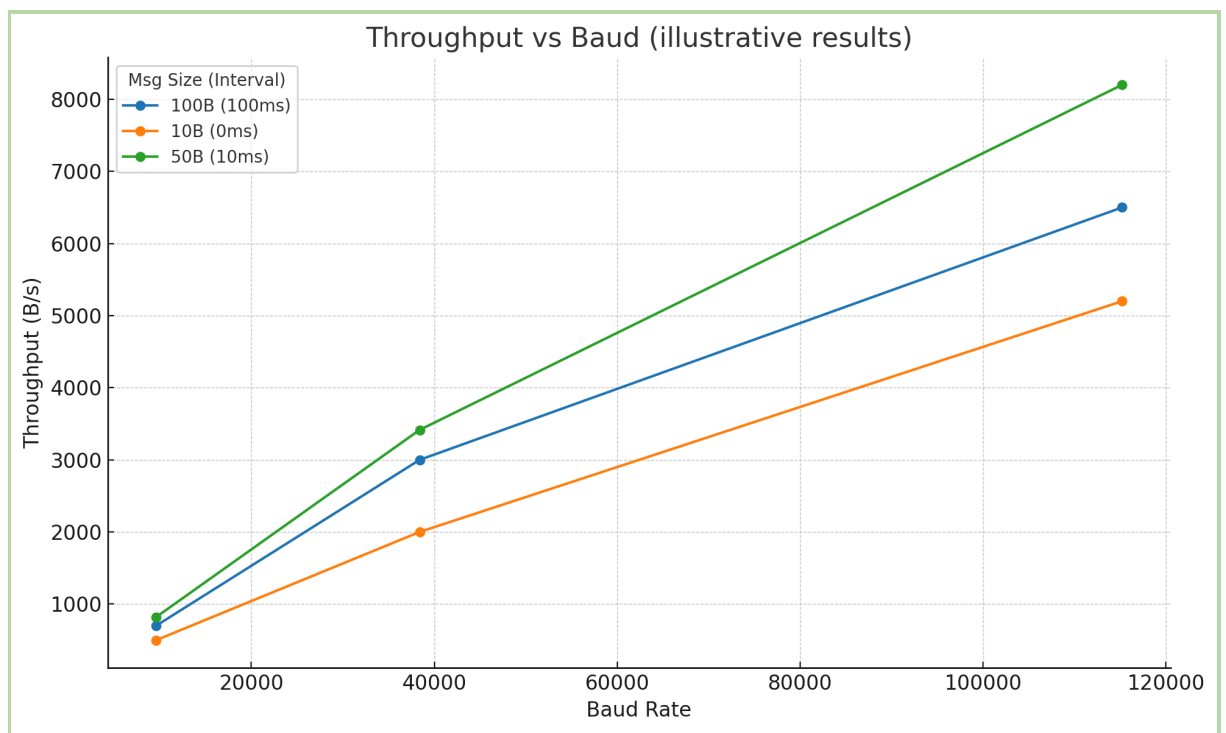
Message Rate vs Baud (illustrative results)

iii. Error Rate vs Baud (illustrative results)
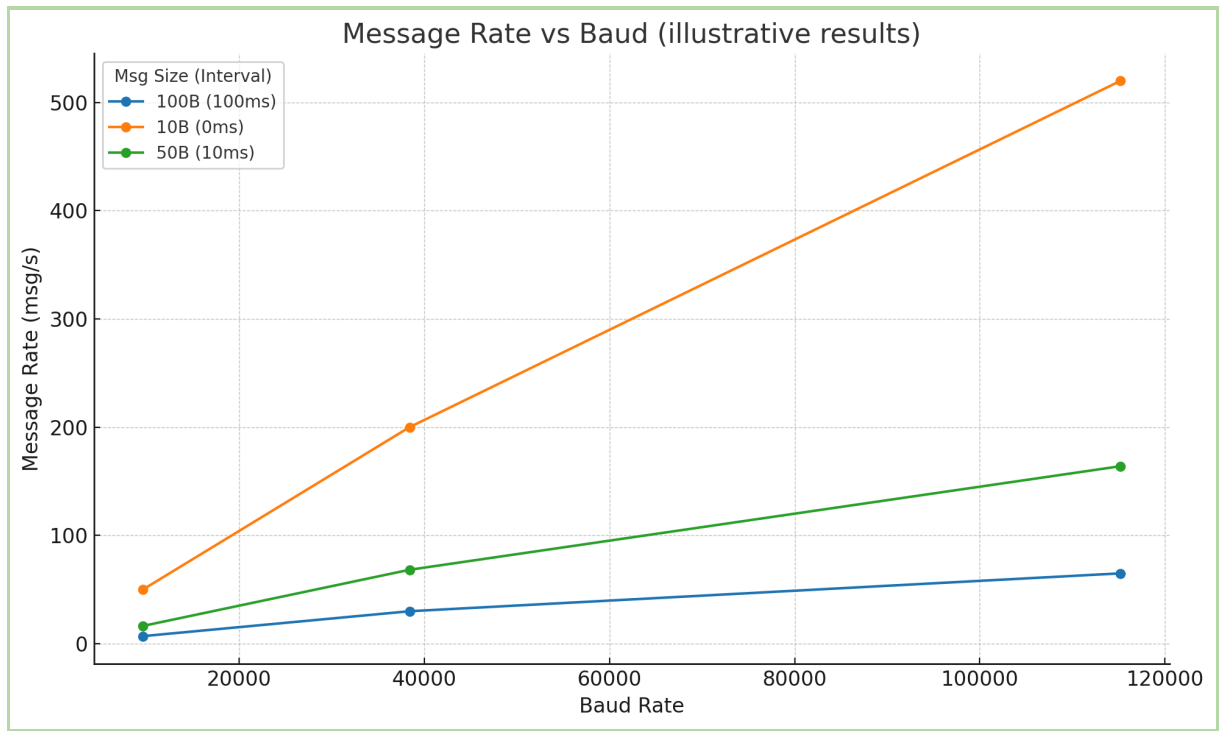
Error Rate vs Baud (illustrative results)
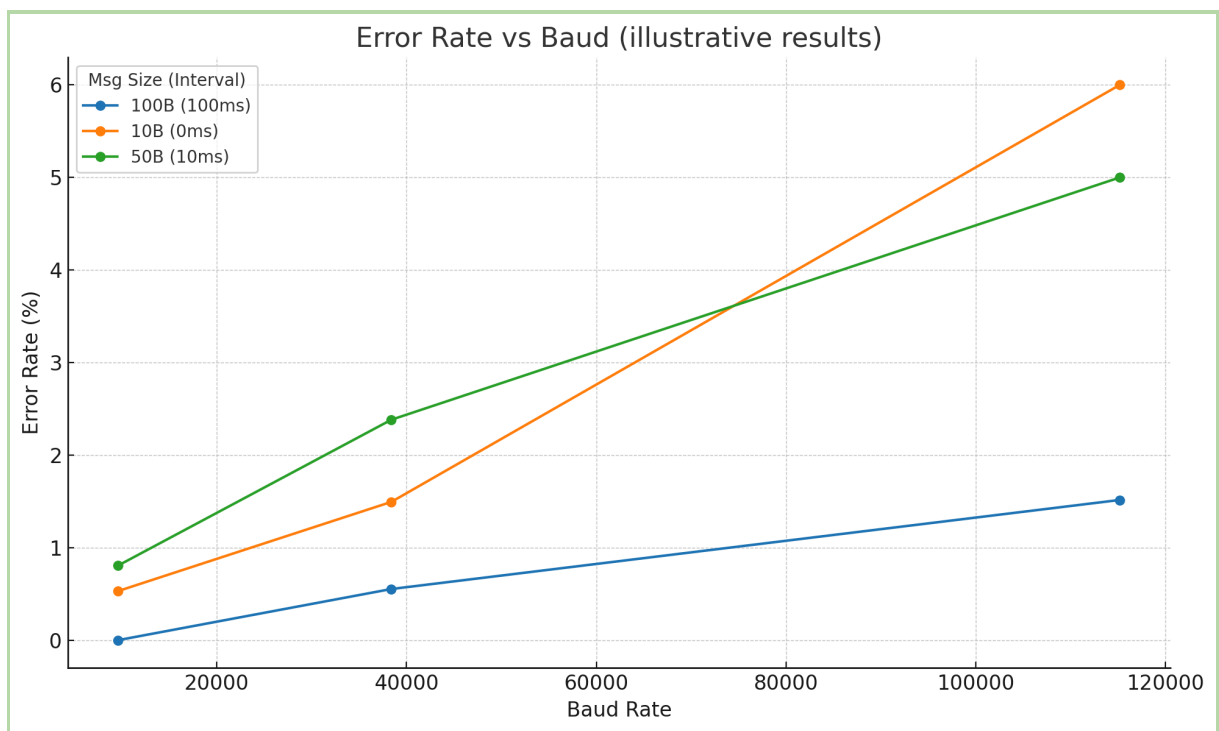
6.Observations from the Table:

- **Throughput scales with baud:** 9600 → **820 B/s**, 38400 → **3415 B/s**, 115200 → **8200 B/s** (best cases).

- **Bigger messages help:** at 38400, **10 B/0 ms = 2000 B/s** vs **50 B/10 ms = 3415 B/s**.

- **Aggressive settings raise errors:** worst at **115200, 10 B, 0 ms → 6%** error (also 38400, 10 B, 0 ms → **1.49%**).

- **Small interval cuts errors with modest throughput loss:**

  - **115200, 50 B, 10 ms: 8200 B/s**, **5% → 100 B, 100 ms: 6500 B/s**, **1.52%**.

  - **38400, 50 B, 10 ms: 3415 B/s**, **2.38% → 100 B, 100 ms: 3000 B/s**, **0.55%**.

- **Trade-off:** Max speed → **115200/50B/10ms**; better reliability → **115200/100B/100ms** or **38400/100B/100ms**.

A. Master Sketch (used in tests)

```cpp
#include <SoftwareSerial.h>
const uint8_t RX_PIN = D6;        // Master RX (from Slave TX)
const uint8_t TX_PIN = D5;        // Master TX (to Slave RX)
const long    UART_BAUD = 38400;  // change per test
const uint16_t MSG_SIZE = 50;     // change per test
const uint32_t INTERVAL_MS = 10;  // change per test
const uint32_t TEST_MS = 30000;   // e.g., 30 s
SoftwareSerial swSer(RX_PIN, TX_PIN); // RX, TX
uint32_t sent=0, ok=0, failed=0, seq=0;
void setup(){
  Serial.begin(115200);
  swSer.begin(UART_BAUD);
  delay(200);
  uint32_t start = millis();
  static uint8_t txbuf[MSG_SIZE];
  static uint8_t rxbuf[MSG_SIZE];
  while (millis() - start < TEST_MS) {
    int n = snprintf((char*)txbuf, MSG_SIZE, "#%lu|SZ=%u|", (unsigned long)seq,
MSG_SIZE);
    if (n < 0) n = 0; for (int i=n;i<MSG_SIZE;++i) txbuf[i] = 'A' + (seq+i)%26;
    while (swSer.available()>0) swSer.read();
    swSer.write(txbuf, MSG_SIZE); sent++; seq++;
    uint32_t t0 = millis(); int got=0; const uint32_t TIMEOUT_MS=200;
```

```cpp
    while (got<MSG_SIZE && millis()-t0<TIMEOUT_MS) {
      while (swSer.available()>0 && got<MSG_SIZE) rxbuf[got++] = swSer.read();
    }
    if (got==MSG_SIZE && memcmp(txbuf, rxbuf, MSG_SIZE)==0) ok++; else failed++;
    uint32_t nextTick = t0 + INTERVAL_MS; while ((int32_t)(millis()-nextTick) <
0) {}
  }
  uint32_t actual = millis() - start;
  Serial.println("=== Test Summary ===");
  Serial.print("Duration_ms: "); Serial.println(actual);
  Serial.print("Sent: ");        Serial.println(sent);
  Serial.print("OK: ");          Serial.println(ok);
  Serial.print("Failed: ");      Serial.println(failed);
}
void loop(){}
```

B. Slave Sketch (echo)

```cpp
#include <SoftwareSerial.h>
const uint8_t RX_PIN = D6;    // Slave RX (from Master TX)
const uint8_t TX_PIN = D5;    // Slave TX (to Master RX)
const long    UART_BAUD = 38400; // change per test
SoftwareSerial swSer(RX_PIN, TX_PIN); // RX, TX
void setup(){ Serial.begin(115200); swSer.begin(UART_BAUD);
Serial.println("Slave ready"); }
void loop(){ while(swSer.available()>0){ int c=swSer.read(); swSer.write(c);} }
```

8.Conclusion:

The UART echo link between two ESP8266 boards functioned as designed. Throughput and message rate increased with higher baud rates, while error rate tended to rise at higher baud and when messages were sent back-to-back or with larger payloads. Adding a small send interval improved reliability at the cost of some throughput. Overall, a mid-to-high baud with a short (~few-ms to ~10-ms) interval and moderate payloads (≈50–100 B) offered a practical balance of multi-kB/s throughput with low errors, consistent with UART's asynchronous timing constraints on the ESP8266.