🟢 TASK 1 — Plain Text Request/Response (Baseline)

Goal: Understand client–server communication and packet visibility.

Task

- Client sends:

  ```
  add 10 20
  ```

- Server responds:

  ```
  result=30
  ```

Concepts Introduced

- TCP sockets
- Client ↔ Server lifecycle
- Request/response model
- Packet capture in Wireshark

Wireshark Focus

- Follow TCP stream
- Observe payload as ASCII
- Identify request vs response

Why this TASK?

This creates a **mental anchor** before introducing structure.

🟡 TASK 2 — Structured Text Protocol (Field-Based)

Goal: Introduce packet structure and parsing logic.

Packet Format (Text-based)

```
OP=ADD;A=10;B=20;
```

Server Response

```
OP=ADD;A=10;B=20;RES=30;
```

Concepts Introduced

- Key–value fields
- Deterministic parsing
- Field validation
- Protocol discipline

Wireshark Focus

- Identify fields in payload
- Explain delimiters (`;` , `=`)

Student Exercise

- Add support for `SUB` , `MUL` , `DIV`

🔴 TASK 3 — Opcode Enumeration (Protocol Evolution)

Goal: Transition from human-readable to **protocol-efficient design**.

Opcode Mapping

Operation	Opcode
ADD	1
SUB	2

Operation	Opcode
MUL	3
DIV	4

Packet Format (Still Text)

```
OP=1;A=10;B=20;
```

Response

```
OP=1;A=10;B=20;RES=30;
```

Concepts Introduced

- Enumeration
- Protocol versioning mindset
- Forward compatibility

Wireshark Focus

- How opcode reduces payload ambiguity
- Why protocols avoid strings

🔵 TASK 4 — Binary Packet Format (Real Networking)

Goal: Teach how **real protocols** work.

✅ Recommended Binary Packet Format

Request Packet

```
+-----------+------------+------------+------------+
| Opcode(1) | Operand1(4)| Operand2(4)| Checksum(1)|
+-----------+------------+------------+------------+
```

Field	Size	Description
Opcode	1 byte	1=ADD,2=SUB,3=MUL,4=DIV
Operand1	4 bytes	int32
Operand2	4 bytes	int32
Checksum	1 byte	Simple XOR or sum

📦 **Total: 10 bytes**

Response Packet

```
+-----------+------------+------------+------------+
| Opcode(1) | Operand1(4)| Operand2(4)| Result(4)  |
+-----------+------------+------------+------------+
```

📦 **Total: 13 bytes**

Concepts Introduced

- Endianness
- Serialization (`struct.pack`)
- Fixed-length packets
- Binary parsing
- Validation

Wireshark Focus

- View packet as hex
- Identify byte boundaries
- Decode integers manually

🔴 TASK 5 — Robust Protocol (Production Thinking)

Goal: Think like protocol designers.

Enhanced Packet Format

```
+--------+--------+-----------+------------+------------+----------+
| Magic  | Ver(1) | Opcode(1) | Operand1(4)| Operand2(4)| CRC(2)   |
+--------+--------+-----------+------------+------------+----------+
```

Concepts Introduced

- Protocol identification (`Magic`)
- Versioning
- Error detection
- Backward compatibility
- Malformed packet handling

Wireshark Focus

- Protocol recognition
- Filters (custom dissector idea)
- Corrupted packet detection

🧪 Suggested Student Challenges

- Write a **packet parser**

- Intentionally corrupt packets
- Add a new operation (MOD)
- Implement protocol version 2
- Measure payload size difference between text vs binary

🛠️ Tools & Stack Recommendation

- Language: **C / Python**
- Libraries:
  - `socket`
  - `struct`
- Capture:
  - Wireshark (`tcp.port == XXXX`)
- Optional:
  - Custom Wireshark dissector (Lua – advanced)