Absolutely! Now that your project includes:

* Reliable UDP with packet loss simulation
* Stop-and-Wait with windowed buffering
* Simulated packet duplication
* HTTP 1.0 support for GET and POST
* A client and server using the protocol

…you should test its robustness and correctness across several cases.

📦 Test Categories

We'll organize tests based on functionality:

1. Basic Functional Tests
2. Packet Loss Simulation
3. Duplicate Packet Handling
4. HTTP Method Parsing (GET and POST)
5. Out-of-Order Packet Handling (windowed protocol)
6. Nonexistent Resources (404)
7. Multiple Sequential Requests
8. Malformed HTTP Requests

🧪 Test Cases

Test 1 – Basic GET Request

Description: Send a well-formed GET request from client to server.

Expected Output:

Client prints:

Sending GET request...  
[Response]  
HTTP/1.0 200 OK  
Content-Length: ...  
Content-Type: text/html

Server prints:

Server ready...  
[Request]  
GET / HTTP/1.0  
Host: 127.0.0.1  
...

Test 2 – Basic POST Request

Description: Send a POST request with a short payload (e.g. "Hello, Server!")

Expected Output:

Client:

Sending POST request...  
[Response]  
HTTP/1.0 200 OK  
Content-Length: 13  
Content-Type: text/plain

Data received

Server:

[Request]  
POST /submit HTTP/1.0  
...  
Received POST data: Hello, Server!

Test 3 – Simulated Packet Loss

Description: Set loss\_rate=0.3 in ReliableUDP and run GET request again.

Expected Output:

Client console may show: timeout, waiting for retransmission  
Server may show: duplicate packets  
But eventually, the request succeeds with the same output as Test 1.

Test 4 – Simulated Packet Duplication

Description: Use loss\_rate=0.15 so that some packets are duplicated.

Expected Output:

Server should receive some "[DUP]" messages and "[RECV]" twice for same seq but should still only ACK once and not duplicate response.

Final response should still be correct.

Test 5 – 404 GET Request

Modify client:

request = (  
"GET /doesnotexist HTTP/1.0\r\n"  
"Host: 127.0.0.1\r\n"  
"User-Agent: ReliableClient\r\n\r\n"  
)

Expected Output:

Client:

[Response]  
HTTP/1.0 404 Not Found  
Content-Length: 0

Server:

[Request]  
GET /doesnotexist HTTP/1.0

Test 6 – Malformed HTTP Request

Modify client:

request = (  
"BADREQUEST / HTTP/1.0\r\n"  
"Host: 127.0.0.1\r\n\r\n"  
)

Expected Output:

Server sends:

HTTP/1.0 404 Not Found  
Content-Length: 0

(you could optionally improve this with a 400 Bad Request message)

Test 7 – Multiple Sequential Requests

Run multiple GETs and POSTs back to back in client.py

Expected Output:

Each request completes reliably. Server prints multiple [Request] logs, client prints correct HTTP responses each time.

Test 8 – Out-of-Order Handling (via Buffer)

Send multiple POSTs quickly, artificially increase WINDOW\_SIZE (e.g., 10), and observe if out-of-order packets are handled.

Expected Output:

Even if packets arrive slightly out of order, the full POST body is assembled and returned. ACKs for each seq# are printed.

Test 9 – Long POST Body

Create a long POST request (e.g., 1024+ bytes of content).

Expected Output:

If your protocol splits or handles packet reassembly correctly, the entire body is received and acknowledged. If not, you’ll see fragmentation issues (a good debugging point).

Test 10 – No ACK Sent (by commenting out server's ack line)

Comment this line in reliable\_recv():

**self.server.sendto(ack\_pkt, adr)**

Expected Output:

Client prints repeated "timeout, waiting for retransmission". This verifies your timeout + retransmission logic works.

✅ Summary Table

| **Test #** | **What it Checks** | **Expected Behavior** |
| --- | --- | --- |
| 1 | Basic GET request | Returns HTML with 200 OK |
| 2 | Basic POST request | Server echoes data, 200 OK |
| 3 | Packet loss simulation | Retransmit on timeout; still succeeds |
| 4 | Packet duplication | Handles duplicated packets gracefully |
| 5 | 404 for unknown path | Server returns HTTP/1.0 404 Not Found |
| 6 | Malformed request | Server handles it with fallback 404 |
| 7 | Back-to-back requests | Reliable, independent responses |
| 8 | Window & reordering support | Server buffers & returns in-order data |
| 9 | Long POST body | Assembles all segments if implemented |
| 10 | ACK dropped | Triggers retransmission |

Would you like me to generate a Python unittest or Pytest file to automate some of these?