Technical Report: Unreliable HTTP Server and Client

Introduction

This technical report provides an overview of the implementation and testing of an unreliable HTTP server and its client. The server handles HTTP requests and returns responses based on a probability distribution. It logs each request's timestamp, client IP address, and outcome. The client interacts with the server to test these responses. The goal of this report is to showcase the server-client communication and present the probability distribution of server responses.

1. Server Code

The server is implemented as a multithreaded HTTP server using Python's built-in libraries. The server handles two main routes: `/getbalance` and `/getlogs`. At `/getbalance`, it randomly chooses between returning a 200 OK response with a fake balance, or simulating an error (403, 500) or timeout. The `/getlogs` route returns a log of all events, including timestamps, client IP addresses, and the server's response outcomes as json format.

1.1 do_GET Function

The `do_GET` function handles incoming GET requests by routing the request to either the `/getbalance` or `/getlogs` handlers. If the requested path is unrecognized, the server returns a 404 error.

```python
LOGFILE = 'server.log'

# Define a multithreaded HTTP server class
class ThreadingHTTPServer(socketserver.ThreadingMixIn, http.server.HTTPServer):
    # Inherits from ThreadingMixIn to handle each request in a separate thread
    pass

# Define a request handler class to handle HTTP requests
class MyHandler(http.server.BaseHTTPRequestHandler):

    # Override the GET request handler
    def do_GET(self):
        # Check the requested path and route accordingly
        if self.path == '/getbalance':
            # Handle the /getbalance endpoint
            self.handle_getbalance()
        elif self.path == '/getlogs':
            # Handle the /getlogs endpoint
            self.handle_getlogs()
        else:
            # If the path is unrecognized, send a 404 error response
            self.send_error(404, "File not found")
```

1.2 handle_get_balance Function

This function is responsible for handling the `/getbalance` endpoint. It randomly chooses an outcome based on predefined probabilities: 20% chance of a timeout, 20% for a 403 error, 10% for a 500 error, and 50% for a successful 200 OK response.

For timeouts, the server sleeps for 15 seconds to simulate a delay. For successful requests, the server responds with an HTML page displaying a fake balance. All events, including client IP, timestamp, and outcome, are logged.

```
# Method to handle the /getbalance endpoint
def handle_getbalance(self):
    # Obtain the client's IP address
    client_ip = self.client_address[0]
    # Get the current timestamp in ISO format
    timestamp = datetime.now().isoformat()
    outcome = ''
    # Generate a random number between 0 and 1 to determine the outcome
    r = random.random()
    # 20% chance to simulate a timeout
    if r < 0.2:
        outcome = 'timeout'
        # Log the event before simulating the timeout
        self.log_event(timestamp, client_ip, outcome)
        # Sleep for 15 seconds to simulate a server timeout
        time.sleep(15)
        # Do not send any response, effectively causing a timeout on the client side
        return
    # 20% chance to return a 403 Forbidden status code
    elif r < 0.4:
        outcome = '403'
        # Send a 403 Forbidden response to the client
        self.send_response(403)
        self.end_headers()
        # Write 'Forbidden' message to the response body
```

```
    # 10% chance to return a 500 Internal Server Error status code
    elif r < 0.5:
        outcome = '500'
        # Send a 500 Internal Server Error response to the client
        self.send_response(500)
        self.end_headers()
        # Write 'Internal Server Error' message to the response body
        self.wfile.write(b'Internal Server Error')
    # 50% chance to return a 200 OK status code with fake content
    else:
        outcome = '200'
        # Send a 200 OK response to the client
        self.send_response(200)
        # Specify that the content type of the response is HTML
        self.send_header('Content-type', 'text/html')
        self.end_headers()
        content = '''
        <html>
        <head><title>Balance</title></head>
        <body>
        <h1>Your Balance</h1>
        <p>$9999</p>
        </body>
        </html>
        '''
```

1.3 run Function

The `run` function sets up the HTTP server and binds it to the specified port (8080 by default). The server runs indefinitely, listening for incoming requests and handling each one in a separate thread.

```
# Function to start the HTTP server on the desired port (default is 8080)
def run(server_class=ThreadingHTTPServer, handler_class=MyHandler, port=8080):
    # Create the server address, binding to all available network interfaces
    server_address = ('', port)
    # Create an instance of the HTTP server with the specified handler
    httpd = server_class(server_address, handler_class)
    print(f'Starting HTTP server on port {port}...')
    # Start serving requests indefinitely
    httpd.serve_forever()
```

1.4 handle_getlogs Function

This function handles the `/getlogs` route. It reads the server's log file and returns the data in JSON format. If the log file cannot be read, the server returns a 500 Internal Server Error.

```
    # Method to handle the /getlogs endpoint
    def handle_getlogs(self):
        # Attempt to read the log file and send its contents to the client
        try:
            with open(LOGFILE, 'r') as f:
                logs = [json.loads(line) for line in f]
            # Send a 200 OK response to the client
            self.send_response(200)
            # Specify that the content type of the response is JSON
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            # Write the logs as a JSON array to the response body
            self.wfile.write(json.dumps(logs).encode('utf-8'))
        except Exception as e:
            # If an error occurs send a 500 Internal Server Error response
            self.send_response(500)
            self.end_headers()
            # Write an error message to the response body
            self.wfile.write(b'Error reading log file')
```

1.5 log_event Function

The `log_event` function records each event in the server log. It logs the timestamp, client IP address, and the outcome of the request. The logs are stored in JSON format, making them easy to retrieve and parse when `/getlogs` is called.

```
    # Function to log events by appending them to the log file in JSON format
    def log_event(self, timestamp, client_ip, outcome):
        with open(LOGFILE, 'a') as f:
            # Create a dictionary representing the log entry
            log_entry = {'timestamp': timestamp, 'client_ip': client_ip, 'outcome': outcome}
            # Write the JSON representation of the log entry to the file, followed by a newline
            f.write(json.dumps(log_entry) + '\n')
```

2. Client Code

The client is responsible for testing the server by making multiple requests to the `/getbalance` and `/getlogs` endpoints. It uses the `requests` library to send HTTP GET requests and displays the response from the server.

2.1 main Function

The `main` function parses the server address and port from the command-line arguments. It constructs the base URL to be used for making requests to the server. The `main` function also initializes a persistent HTTP session and defines the number of requests that will be made to the `/getbalance` endpoint.

```python
def main():
    # Check if the user provided exactly two command-line arguments (excluding the script n
    if len(sys.argv) != 3:
        # If not, display the correct usage and exit the program
        print('Usage: python client.py <server_address> <port>')
        sys.exit(1)

    # Extract the server address and port number from the command-line arguments
    server_address = sys.argv[1]
    port = sys.argv[2]

    # Construct the base URL using the provided server address and port number
    base_url = f'http://{server_address}:{port}'

    # Create a Session object to persist certain parameters across requests
    # and to reuse the underlying TCP connection for multiple requests
    session = requests.Session()

    # Define the number of times to call the /getbalance endpoint
    num_requests = 20
```

```python
# Loop to send multiple GET requests to the /getbalance endpoint
for i in range(num_requests):
    try:
        # Send a GET request to the /getbalance endpoint with a timeout of 10 seconds
        response = session.get(f'{base_url}/getbalance', timeout=10)

        # Print the status code received from the server
        print(f'/getbalance response status code: {response.status_code}')

    except requests.exceptions.Timeout:
        # Handle the case where the request times out
        print('/getbalance request timed out')

    except requests.exceptions.RequestException as e:
        print(f'/getbalance request failed: {e}')

    # Wait for half a second before sending the next request to avoid overwhelming the server
    time.sleep(0.5)
```

2.2 GET Requests to /getbalance

The client sends multiple GET requests to the `/getbalance` endpoint in a loop. For each request, it prints the response status code. If a request times out or fails, the client handles the exception and prints an error message. After each request, the client waits for half a second before sending the next one to avoid overwhelming the server.

```python
    # After completing the /getbalance requests, retrieve the server logs
    try:
        response = session.get(f'{base_url}/getlogs')
        print('Logs:')
        print(response.text)

    except requests.exceptions.RequestException as e:
        # Handle any exceptions that occur while requesting the logs
        print(f'/getlogs request failed: {e}')

if __name__ == '__main__':
    main()
```

2.3 GET Request to /getlogs

After sending the `/getbalance` requests, the client retrieves the server logs by sending a GET request to the `/getlogs` endpoint. The response contains the log data, including the client IP addresses, timestamps, and outcomes of the server requests.

```python
if __name__ == '__main__':
    from sys import argv
    # If a port number is provided as a command-line argument
    if len(argv) == 2:
        # Run the server on the specified port
        run(port=int(argv[1]))
    else:
        # Otherwise, run the server on the default port (8080)
        run()
```

3. Implementation and Testing

To demonstrate the server-client interaction, the following images show the server's IP address, client IP, and client usage.

Server IP Address:

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=6463<RXCSUM,TXCSUM,TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_
CSUM>
        ether a4:83:e7:22:db:0a
        inet6 fe80::816:6c32:f352:6840%en0 prefixlen 64 secured scopeid 0x6
        inet 10.0.28.182 netmask 0xffff0000 broadcast 10.0.255.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
        status: active
```

Client IP Address:

```
        status: inactive
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        options=6460<TSO4,TSO6,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
        ether f6:0b:6f:35:36:22
        inet6 fe80::14a1:cc3f:3345:b449%en0 prefixlen 64 secured scopeid 0xb
        inet 10.0.142.133 netmask 0xffff0000 broadcast 10.0.255.255
        nd6 options=201<PERFORMNUD,DAD>
        media: autoselect
```

Client Usage Output:

```
(base) aahmadli14270@MacBook-Air--Ali-4 Desktop % python3 client.py
Usage: python client.py <server_address> <port>
(base) aahmadli14270@MacBook-Air--Ali-4 Desktop % python3 client.py 10.0.28.182 8080
/getbalance request timed out
/getbalance response status code: 403
/getbalance response status code: 200
/getbalance request timed out
/getbalance response status code: 200
/getbalance response status code: 200
/getbalance response status code: 200
/getbalance request timed out
/getbalance request timed out
/getbalance request timed out
/getbalance response status code: 200
/getbalance response status code: 200
/getbalance request timed out
/getbalance response status code: 200
/getbalance response status code: 403
/getbalance response status code: 200
/getbalance response status code: 200
/getbalance response status code: 500
/getbalance response status code: 200
/getbalance response status code: 500
```

Client Response Logs: