Simulating IoT Backend Systems Device Communication via Node.js and WebSockets

Alperen Kayhan
alperen-1393@hotmail.com
https://github.com/AlperenKayhan

***Abstract*—This paper presents a Node.js-based simulator that mimics a IoT Backend Systems-enabled embedded device. It performs session initialization, WebSocket connection setup, custom logging, heartbeat-based liveness detection, and remote command handling with simulated faults via Poisson-distributed failures. The project demonstrates a robust integration testing tool for IoT Backend Systems server endpoints and message flows.**

I. Introduction

The IoT Backend Systems system is an industrial-grade platform for controlling and monitoring embedded devices. Typically implemented in C/C++, Python and JavaScript; and deployed on physical hardware, testing and simulation of its behavior can be challenging. This project replaces the hardware simulation with a pure Node.js script that communicates with the IoT Backend Systems server backend, mimicking the real-time behavior of a deployed device.

II. Simulation Device

Test device are consist of raspberry Pis which, they were run on Linux based OS. To run our scripts we may sended necessary scripts or get help tools WinSCP.

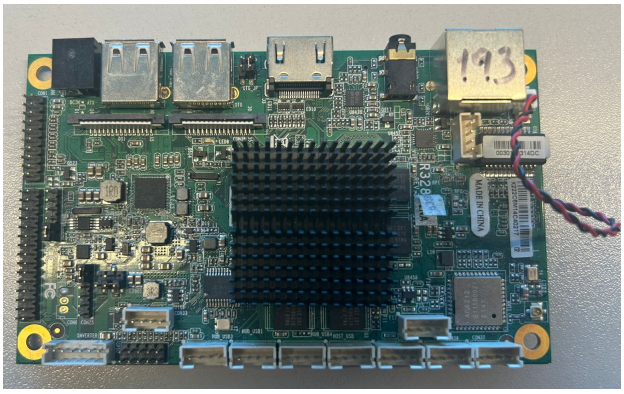

Fig. 1. Figure of Test Device

III. System Architecture

The proposed Node.js client follows these stages:

1) Builds the `DvOp` URL with all parameters (MAC, IP, Serial, etc.).
2) Sends an HTTP GET request to initialize a session.
3) Extracts the session ID from the response.
4) Establishes a WebSocket connection using Socket.IO with a custom cookie path.
5) Registers listeners for different commands such as reboot, log messages, or device queries.

Manuscript received July 4, 2025.

IV. functions

In this part I will be explaing our functions that is used in our script.

A. *buildURL()*

This function constructs a valid IoT Backend Systems service URL by injecting runtime-specific parameters such as device MAC, IP address, firmware version, and serial number into the query string.

B. *getNetworkInfo()*

It extracts the local network interface information, selecting the first active non-internal IPv4 address and its associated MAC, simulating how a real device identifies itself on the network.

C. *poissonRandom()*

Implements the Poisson process to model random hardware fault intervals. Used to probabilistically trigger auto-reboot events during operation.

D. *fetchSession(serialNo)*

Contacts the IoT Backend Systems backend with the device serial number, parses the JSON response, and retrieves the session ID along with other identifiers necessary for WebSocket authentication.

E. *loadSessionId()*

Reads a previously saved session ID from the `sessionID.txt` file. If the file is missing or empty, it terminates the process to avoid sending unauthenticated requests.

F. *uploadLogFile(sessionId, serialNo)*

Takes a snapshot of the current JSON-formatted log file and uploads it as a multipart form to the backend via HTTP POST. A temporary copy is used to preserve the active log.

G. startClient(sessionId)

Initializes the Socket.IO WebSocket connection using the given session ID, attaches relevant event handlers, and begins the heartbeat timer to maintain presence.

V. FUNCTIONS

This part explaining our case condition that we are using in our design. With help of the payload ”payload = JSON.parse(raw.t);” element; we can listen to IoT Backend Systems messages that we received. Thus, with the listened message we can run the desired case conditions.

A. sendmsglog

Displays textual logs pushed from the server to the console, mirroring UI-side messages during runtime diagnostics.

B. reboot

Triggers an internal client reset by restarting the script process itself, without affecting the operating system or host device.

C. PowerOff

Exits the Node.js process gracefully, emulating a complete device shutdown initiated by the backend.

D. get_d_parameters

Collects and prints key runtime parameters like MAC address, local IP, and site-specific IDs, useful for debugging or verification.

E. change_d_parameters

Activates the Poisson-based fault injection mechanism, allowing the next heartbeat event to probabilistically trigger a simulated reboot.

F. get_d_items

Logs incoming payloads along with device identity info to a persistent file, enabling forensic audit trails and offline analysis.

G. send_logs

Uploads the current log data to the IoT Backend Systems backend. After successful transmission, the temporary file copy is deleted to free resources and prevent data duplication.

H. (default)

Any unexpected message types are printed to the console for transparency and debugging, allowing for future protocol expansion.

VI. METHODOLOGY

A. Heartbeat Mechanism

A custom ping is sent every 5 seconds to simulate liveness. This ping cycle is also used to trigger fault simulation via a Poisson process:

$$P(k;\lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{1}$$

where $\lambda = 5$ in our implementation. When $k \leq 2$, an auto-reboot is triggered.

B. Command Handlers

Table I lists all supported messages received via WebSocket.

TABLE I
WEBSOCKET MESSAGE HANDLERS

Command	Action
send_msg_log	Print and log the message
reboot	Restart the client script
get_d_items	Log network parameters
changed_parameters	Trigger fault simulation
Power_Off	Cleanly exit the process

VII. RESULTS

The client runs continuously and correctly interacts with the IoT Backend Systems backend. When remote UI buttons are triggered (e.g., reboot, send message), appropriate client-side behaviors are observed and verified via the console and log file.

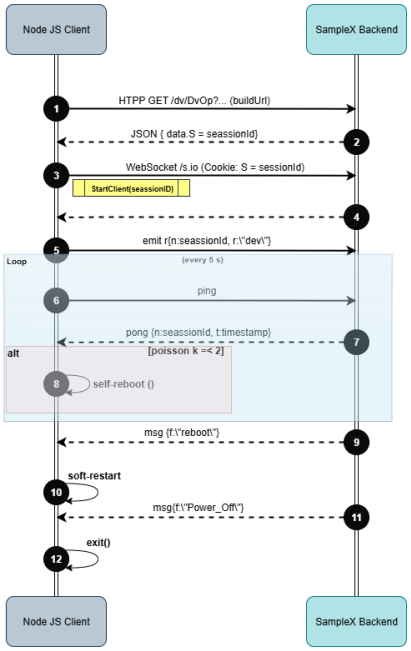

Fig. 2. Sequence Diagram

VIII. CONCLUSION

This Node.js client offers a flexible and maintainable way to simulate IoT Backend Systems devices without actual hardware. With the heartbeat-based Poisson error injector and full command support, the client is suitable for testing, debugging, and demonstrations of IoT Backend Systems systems.

ACKNOWLEDGMENT

I would like to thank the IoT Backend Systems engineering team and thier support during the development of this project.

REFERENCES

Khan, M. T. R. (2025). Discrete-Event Simulation [Lecture slides]. CNG-476 System Simulation, Computer Engineering Department, Middle East Technical University, Northern Cyprus Campus.