August 2023

Visual Debugger for Internet-of-Things (IoT) Ecosystem

Sean Zarringhalam

Visual Debugger for Internet-of-Things (IoT) Ecosystem

ABSTRACT

When a smart home device does not work as expected, the device vendor has to investigate many interacting parts in the ecosystem in order to determine a root cause for the issue. With a high number of interacting parts involved in debugging, the debugging process can be time-consuming and tedious. This disclosure describes a visual debugger that is optimized for IoT ecosystems. The visual debugger unifies logs generated by IoT devices to display the evolution of device states with time. Network requests and responses are indicated visually by clickable arrows indicating the directions of network traffic. An event on a device that triggers a cascade of actions on other devices is displayed using an action path that indicates the movement of the action through the network. A log file framework enables components of the IoT network originating from different manufacturers to be analyzed and displayed in a systematic and unified manner. IoT network developers and third-party device vendors can efficiently triage bugs, determine the root causes of failures, and improve performance and interoperability.

KEYWORDS

- Internet of things (IoT)
- Visual debugger
- Smart home device
- Border router
- Log processing
- Replay mode debugging
- Live mode debugging

BACKGROUND

When a smart home device does not work as expected, the device vendor has to investigate many interacting parts in the ecosystem in order to determine a root cause for the issue. For example, the smart home device may have lost internet connectivity for a brief period of time; the device may have a hardware error; there may have been an error in the user interface used to control the device; etc. With a high number of interacting parts involved in debugging, the debugging process can be time-consuming and tedious.

Although tools exist that can trace events across services, logs of such tools typically represent one request/response across multiple services. The problem of understanding the state of multiple devices in one ecosystem (e.g., a home network or structure) at a specific point in time remains unsolved. This is especially an issue with Internet-of-Things (IoT) protocols that enable devices to directly talk to each other rather than using cloud-based services for device control.

DESCRIPTION

This disclosure describes a visual debugger optimized for Internet-of-Things (IoT) ecosystems. The visual debugger unifies logs generated by IoT devices on a network to display the evolution of device states with time. Network requests and responses are indicated visually by clickable arrows indicating the direction of network traffic. Clicking on an arrow displays the contents of network traffic. An event on a device that triggers a cascade of actions on other devices is displayed using an action path that indicates the movement with time of the action through the network. A log file framework enables components of the IoT network that originate from different manufacturers to be analyzed and displayed in a systematic, unified, and visual manner.

A typical use case for the described visual debugger for IoT ecosystems is as follows. A user has set up a home automation that turns on a smart light in their living room whenever the smart lock for their front door is unlocked. The smart lock communicates to the smart light directly via a local network rather than via the cloud. Although the user has set up this automation correctly, for some reason, the smart light does not turn on when the smart lock is unlocked. The user reports the incident in their mobile app that is used to control the smart lock. The smart lock manufacturer receives the feedback and uses the visual debugger to troubleshoot.

The visual debugger can operate in two modes: replay and live. The replay mode can be used to debug an issue that has occurred in the past but can be debugged via log files that were uploaded after a user reports the bug. The live mode can be used to debug an issue as it happens, e.g., the state of the visual debugger updates in real time based on the state of the devices.

Log files

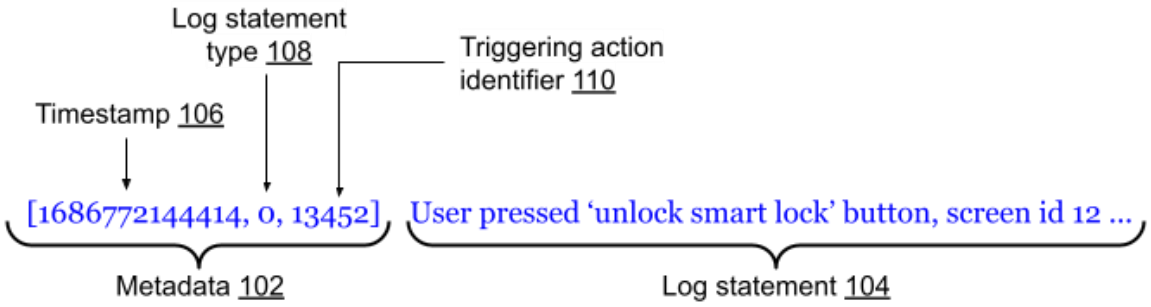

Fig. 1: Example line of a log file

The visual debugger relies on log files generated by the device to display the states of devices. The log files comply with the following format, illustrated in Fig. 1.

- Lines in the log files have metadata (102, which can be indicated as such, e.g., using square brackets) and a log statement (104), such that a line in the log file reads as: [metadata] log statement.

- The metadata section includes the following values (e.g., in comma-separated format):
    - Timestamp (106): A number representing microseconds in Coordinated Universal Time (UTC) when the event occurred.
    - Log statement type (108): A numerical enumerator representing the type of log statement. For example, a log statement can either be a triggering action (0), a network request (1), a trait update (2), or other behavior (3).
    - Triggering action identifier (110): A unique string representing the action that triggered the behavior, which may have occurred on a different device. For example, in the example where the unlocking of the smart lock triggers the turning on of the light, the action identifier can refer to the button-press action on the controlling app labeled ‘unlock smart lock.’
- The contents of the log statement depend on the log statement type, as follows:
    - Where the log statement type is a triggering action, the log statement includes a description of that triggering action. For example, a user pressing a “unlock smart lock” button on their mobile app can look thus (triggering action is underlined for clarity): [1686772144414, 0, 13452] User pressed ‘unlock smart lock’ button, screen id 123, element id 345.
    - Where the log statement type is a network request, the log statement includes the contents of that network request in a standard format, e.g., JavaScript Object Notation (JSON) format. For example, if an HTTP request is sent to a server in response to the “unlock smart lock” button being pressed, the log statement may be (network request is underlined for clarity): [1686772144420, 1, 13452] {RequestUrl: “www.examplesite.com/serverUrl”, “method”: “POST”, ... }.

- Where the log statement type is a trait update, the log statement includes a trait update message following a standard JSON format. For example, if the smart lock changes to the "locked" state, the log entry may be (trait update is underlined for clarity): [1686772144430, 2, 13452] {trait: LockUnlock, value: "{isLocked: true}"}.
- Where the log statement type is other behavior, the log statement can be a string description of the event that occurred. For example, if the smart lock unlocking mechanism takes 5 seconds, the corresponding log statement may be (description is underlined for clarity): [1686772144450, 3, 13452] Unlock mechanism took 5 seconds.

Log processing

Log processing is illustrated in Fig. 2. When a user (202) encounters an issue (204) and submits feedback (206) in their home control (IoT) app, devices (208) in the user's network are triggered (210) to upload log files (212) to a cloud service, referred to as a log-collection service (214). The log files are uploaded from a start time that is specified when initiating feedback. The log files are stored in a central database keyed by a feedback session identity, such that specific feedback session data can be retrieved at a later time. Log files thus stored can be used to support replay mode.

The visual debugger (218, run by a vendor/developer) receives the user's feedback report (220) and presents it in a feedback dashboard. The developer can click on the report to view it. A button "start visual debugger" (or similar) can be clicked to start the visual debugging session (222). Upon starting the visual debugging session, log files are received (224) by the visual

debugger and used for the purposes of analysis and graphical display, described in greater detail below.

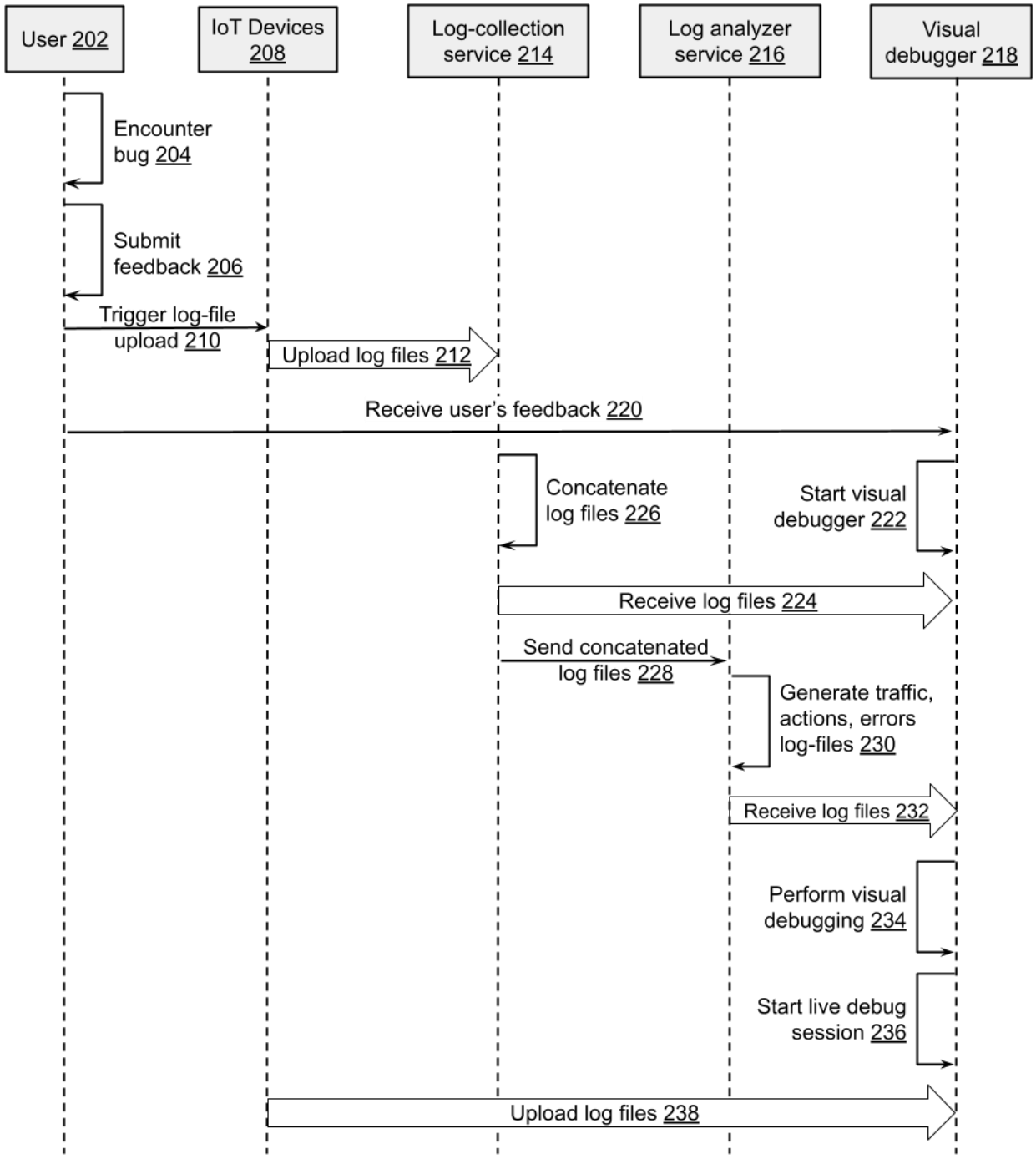

Fig. 2: Log processing

When a feedback session is sent to the log collection service, the log collection service concatenates the log files (226) together into a single log file and sends the concatenated file (228) to a log analyzer service (216). Each entry in the concatenated log file is appended with an additional metadata field that includes the device identity of the entry. The log analyzer service accepts as input a single log file that includes logs for all devices in the network, and generates output log files (230) as follows:

- Network_traffic.log, which includes blocks of network request/response pairs separated by empty newlines. The network_traffic.log file is used to support network request/response indicators (described in greater detail below) in the visual debugger.
- Actions.log, which includes blocks of log statements belonging to the same action trigger, blocks being separated by empty newlines. The actions.log file is used to support the action-path use case (described in greater detail below) in the visual debugger.
- Errors.log, which includes errors that have occurred in the network for the given time-based session.

The network_traffic.log, actions.log, and errors.log files are received (232) by the visual debugger and used for the purposes of analysis and visual debugging (234), described in greater detail below.

To debug an issue in real time, a “start live debug session” button within the visual debugger can be selected (236). Selection of this button causes devices to stream their log files to a computer running the visual debugger (238), e.g., via web sockets over HTTP. The computer running the visual debugger must be on the same network as the IoT devices being debugged to support live mode. When live mode terminates, feedback is effectively already submitted, such that the live mode debug session can be revisited later in replay mode.

Stand-in for border router

When the visual debugger is in live mode, the web browser running the visual debugger instructs devices in the network to treat it as the border router. For devices enabled with device-to-device communication protocols, the browser becomes the border router for the duration of the live-mode debugging session. Rather than relying solely on log files as a trait-update signal, having the browser act as the border router gives the debugger a more accurate trait-update signal. This enables the detection of traits that have successfully propagated from related devices, in turn enabling live inspection view through the visual debugger.

Developer experience

When a developer opens the visual debugger, they are presented with a UI that includes:

- **A web-based smart home control UI:** This is equivalent to what an end user would see to control their home. Devices in the network can be displayed here with their real-time state, as the user would see them.
- **Time-based controls:** These can appear as a scrubber bar (a slider, similar to those found in audio/video playback) at the bottom, which the user can drag along in order to change the time that the visual debugger derives state from. A play/pause button enables the user to watch the changing states and their rate of change.
- **Device logs:** A device appearing on the screen has an expandable icon that can be clicked in order to view the full logs for that device, updating as the progress bar at the bottom of the screen progresses, reflecting the device log at that specified time.
- **Network request/response indicators:** A network request that is sent (and a response that is received) triggers the appearance of arrows indicating the direction of the network

traffic (incoming vs outgoing). Clicking the arrow expands a pane showing the contents of the network traffic.

- **Action path:** When a triggering action is detected on any device (e.g., a button press on a phone, a motion detected in front of a camera), a new line appears in the ‘actions’ window. Clicking that line shows how that action travels through the network. For example, tapping on a line labeled ‘unlock smart-lock button click on mobile app’ action opens a full screen showing the path of the resulting outcomes in chronological order. In the example of the line labeled ‘unlock smart-lock button click on mobile app,’ the action path can be:
    - unlock smart lock button click on mobile app;
    - request sent to smart lock <device id> to unlock;
    - smart lock unlocked.
- **Error notifications:** Devices that log an error immediately have a clickable error icon that appears by the device. Clicking the error icon reveals the error log message in the context of the log file of the device.
- **Live inspection view (live mode only):** Because the web view acts as a border router, it can display the state of the border router in live inspection view. This view shows the state of the network as the border router sees it, and also provides a command line to query this state directly.

The described visual debugging techniques are applicable generally to IoT networks (such as networks of smart-home devices and sensors). In particular, the visual debugger can be provided as a tool to third-party vendors who manufacture or develop various components, devices, and sensors of a network. The visual debugger can enable efficient triaging of bugs and

determination of root causes of failures, leading to improved performance and interoperability. Third-party vendors that adhere to the above-described logging specification can have their devices appear correctly in the visual debugger. A binary can be provided to third-party vendors to run on their devices to handle the uploading of log files to the right destination in the right format under the right triggers (e.g., when the user initiates feedback in a mobile app). In this manner, the developer or integrator of an IoT network can host components from multiple different IoT (smart home) component vendors and enable the systematic, unified, and visual analysis of feedback logs generated by IoT components originating from multiple different vendors.

CONCLUSION

This disclosure describes a visual debugger that is optimized for IoT ecosystems. The visual debugger unifies logs generated by IoT devices to display the evolution of device states with time. Network requests and responses are indicated visually by clickable arrows indicating the directions of network traffic. An event on a device that triggers a cascade of actions on other devices is displayed using an action path that indicates the movement of the action through the network. A log file framework enables components of the IoT network originating from different manufacturers to be analyzed and displayed in a systematic and unified manner. IoT network developers and third-party device vendors can efficiently triage bugs, determine the root causes of failures, and improve performance and interoperability.

REFERENCES

1. Connected Standards Alliance. “Matter, the foundation for connected things,” available online at https://csa-iot.org/all-solutions/matter/ accessed July 28, 2023.

2. “What Is a Thread Border Router? (And Do I Need to Buy One?)” available online at https://www.howtogeek.com/830124/what-is-a-thread-border-router-and-do-i-need-to-buy-one/ accessed August 13, 2023.
3. “Smart Home Device Traits | Cloud-to-cloud | Google Home Developers” available online at https://developers.home.google.com/cloud-to-cloud/traits accessed August 13, 2023.
4. Castillo, Carlos Rojas, Matteo Marra, Jim Bauwens, and Elisa Gonzalez Boix. "Out-of-Things Debugging: A Live Debugging Approach for Internet of Things." *arXiv preprint arXiv:2211.01679* (2022).
5. Rodrigues, Andreia, Jose Pedro Silva, João Pedro Dias, and Hugo Sereno Ferreira. "Multi-Approach Debugging of Industrial IoT Workflows." *arXiv preprint arXiv:2009.05828* (2020).