Using MQTT-over-WebSocket with RAFW Connectivity Devices

This document describes how to use the MQTT Client over WebSocket application on the RA6W1/RA6W2, enabling secure MQTT communication through a WebSocket transport channel with automatic reconnect and MQTT state management. It is applicable to the RRQ61001 and RRQ61051 modules.

Contents

1. Introduction **2**
2. System Overview **2**
2.1 Software Requirements 4
2.2 Hardware Requirements 4
3. MQTT-over-WebSocket Application Setup **5**
3.1 Set Up EK-RA6W1/RA6W2 5
3.2 Set Up MQTT Broker with WebSocket Support 5
3.3 Start MQTT Broker 6
4. MQTT-over-WebSocket Bridge Example **7**
4.1 Configure Application Parameters 7
4.2 Program Firmware Image 7
4.3 Connect Device 8
4.4 Connection Management and MQTT Recovery 11
5. Revision History **12**

Figures

Figure 1. Sequence diagram 3
Figure 2. Mosquitto broker running with both MQTT and WebSocket ports 6
Figure 3. Updating the WebSocket server's IP address 7
Figure 4. Example of the CA certificate format 7
Figure 5. Setting up the device in Station mode 8
Figure 6. Configuring the SNTP Client 8
Figure 7. Console log showing device connected to the AP and SNTP server 9
Figure 8. J-Link terminal connection parameters 9
Figure 9. Receiving published messages on the J-Link terminal 10
Figure 10. Logs on the Mosquitto broker 10
Figure 11. Message sent from PC received by the device 10
Figure 12. Logs on the Mosquitto subscriber terminal 10
Figure 13. J-Link logs indicating disconnect and automatic reconnect 11

© 2026 Renesas Electronics

Terms and Definitions

EVK	Evaluation Kit
RTOS	Real-Time Operating System
AP	Access Point
SSID	Service Set Identifier
UART	Universal Asynchronous Receiver/Transmitter
Wi-Fi®	Wireless Fidelity
WLAN	Wireless Local Area Network

1. Introduction

The MQTT Client over WebSocket Application demonstrates secure MQTT client communication tunneled through a WebSocket transport layer on the Renesas RA6W1/RA6W2 platform. The application establishes a TLS-encrypted WebSocket connection to a Mosquitto MQTT broker configured with a dedicated WebSocket listener, negotiates the MQTT subprotocol, and then performs MQTT connect, subscribe, publish, and keep-alive operations over the secured channel.

2. System Overview

This implementation requires both the RA6W1/RA6W2 device and the Mosquitto broker to be provisioned with matching TLS certificates. The broker is configured with a WebSocket listener on port 8765 that validates client connections using a self-signed or CA-signed server certificate. The device, in turn, is configured with the broker's root CA certificate to verify the server during the TLS handshake.
The system flow is as follows:
By default, the device boots in Station (STA) mode and attempts to connect using previously stored Wi-Fi credentials. If no valid credentials are available, or after a factory reset, the device enters Easy Setup mode and prompts the user for Wi-Fi credentials through the UART console.
After the credentials are entered, the device connects to the Wi-Fi access point in Station mode. The application optionally synchronizes time using SNTP to enable TLS certificate validation.

The MQTT broker is set up with the WebSocket port configured in its `.conf` file and path to server certificates generated by OpenSSL.
A TLS handshake is initiated with the Mosquitto broker on port 8765, and the server certificate is validated against the loaded CA certificate. The WebSocket upgrade request is sent with the MQTT subprotocol negotiation.

The application sends an MQTT CONNECT packet, followed by SUBSCRIBE to configured topics (for example, `rrq61x_sub`). The broker confirms with CONNACK and SUBACK responses.
The device begins publishing MQTT messages periodically to `rrq61x_pub` and receives incoming PUBLISH messages from subscribed topics.

Connection health is maintained via TCP keep-alive and WebSocket ping/pong frames. If the connection is lost, exponential backoff reconnection logic retries the complete sequence.
Upon reconnection, MQTT state is reset and the CONNECT/SUBSCRIBE sequence resumes.

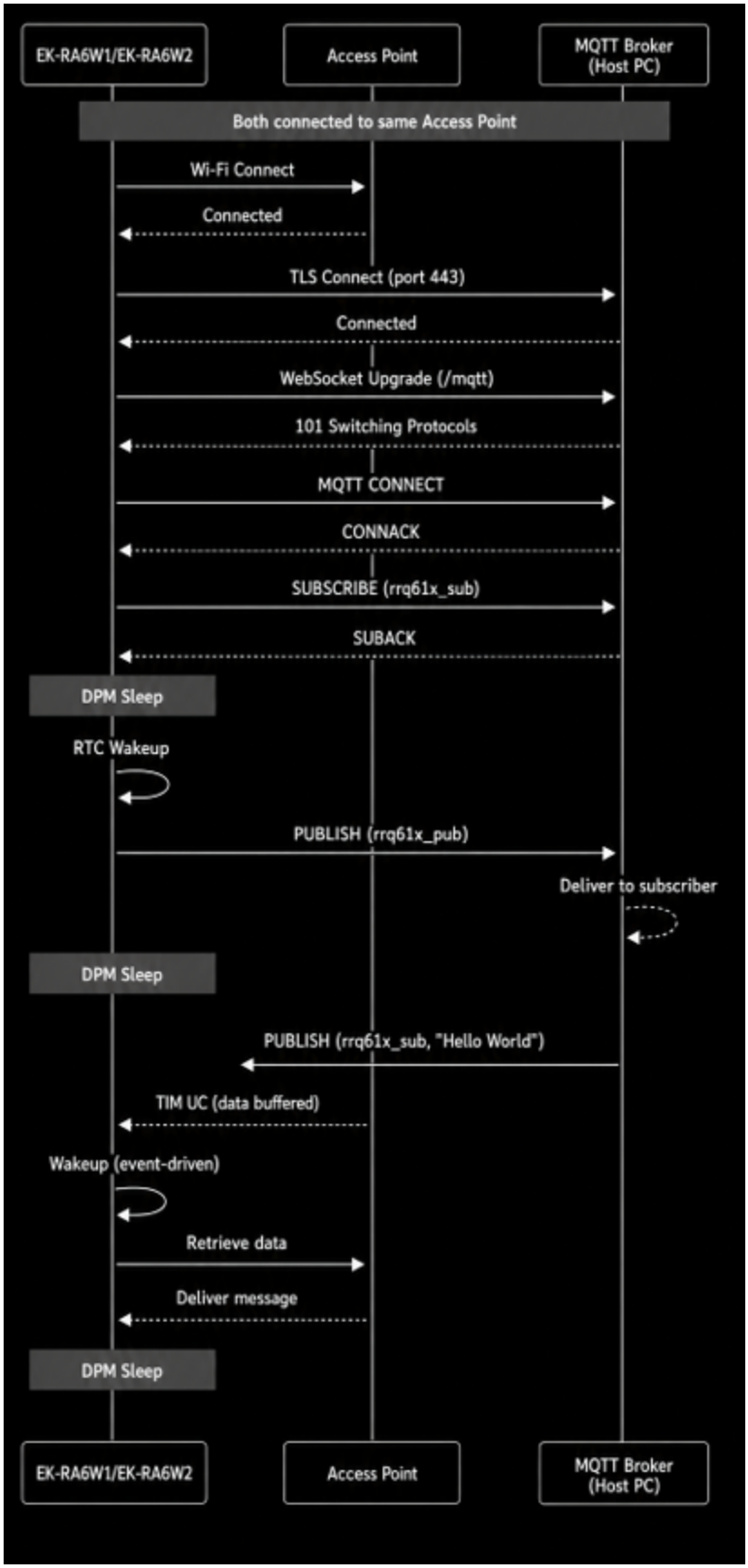


Figure 1. Sequence diagram

2.1 Software Requirements

The following software tools and packages are required to complete the project:

- Renesas Electronics Flexible Software Package (FSP): Version 2.2.0 or later.
- e^2 studio IDE: Version 2026-04.3 or later.
- GCC Arm Embedded Toolchain: Version 13.3.1.arm-13-24.
- Serial Terminal Application: Tera Term or similar tools for logging and monitoring output.

2.2 Hardware Requirements

- Renesas RA6W1/RA6W2 Evaluation Kit (EVK).
- RA6W1/RA6W2 MCUwith ultra-low power consumption.
- USB Type-C Cable: For programming and debugging.
- Jumper Wire (Female to Female): For making proper motherboard configurations for the application.
- Host PC: For running the Mosquitto MQTT broker and test clientsFor programming, viewing console logs and giving AT commandsFor programming, viewing console logsFor programming, viewing console logs, opening MQTT test client and giving AT commands.
- Wi-Fi Access Point (AP): With network connectivity.

3. MQTT-over-WebSocket Application Setup

Before running the MQTT-over-WebSocket application, install the required software, configure the evaluation kit, and set up the MQTT broker. See the following sections for instructions.

3.1 Set Up EK-RA6W1/RA6W2

An RA6W1/RA6W2 Evaluation Kit is required to run the MQTT-over-WebSocket example.
Ensure the following:

- The RA6W1/RA6W2 module is installed on the board
- USB debug port is connected to the host PC
- A Wi-Fi Access Point is available and reachable from the RA6W1/RA6W2 device
- The host PC and device share network connectivity or have routable access to the WebSocket server
- OpenSSL commands are used to generate the server certificate details
- The MQTT broker supports WebSocket transport on the configured port.

3.2 Set Up MQTT Broker with WebSocket Support

Use OpenSSL to generate the TLS server certificates for the server as well as the CA certificate using the provided commands:

```
openssl ecparam -genkey -name prime256v1 -out ca.key
openssl req -x509 -new -key ca.key -out ca.crt -days 3650 -subj "/CN=MyRootCA"
openssl req -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -nodes -keyout server.key -out
server.csr -subj "/CN=<ip_address>"
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days
3650
```

On the system running the MQTT broker, add the path to those certificates as well as the WebSocket listener port used for handling WebSocket connections in the configuration file as follows:

```
listener 8765
protocol websockets

cafile <Path to file>\ ca.crt
certfile <Path to file>\server.crt
keyfile <Path to file>\server.key

# Enable detailed logging to see the WebSocket handshake
log_type all
```

This configures the broker to accept secure WebSocket connections on port 8765 and verify the server certificate chain.

3.3 Start MQTT Broker

To get the MQTT broker up and running, use the following command:

```
mosquitto -c "C:\Program Files\mosquitto\mosquitto.conf" -v
```

The following figure shows the Mosquitto broker with both the WebSocket and the MQTT ports configured.

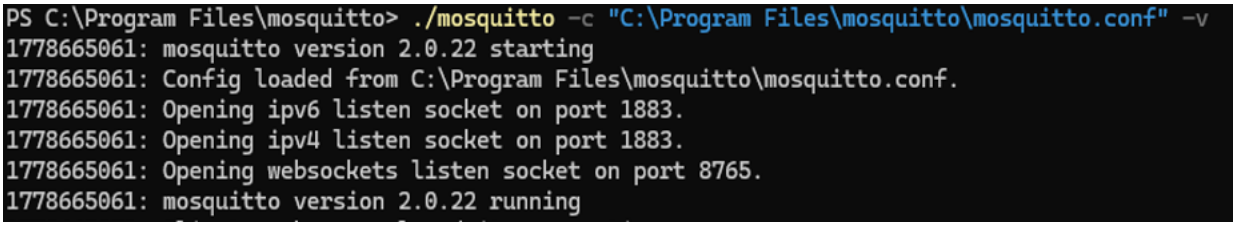


Figure 2. Mosquitto broker running with both MQTT and WebSocket ports

4. MQTT-over-WebSocket Bridge Example

This section describes how to get the MQTT-over-WebSocket application running on the RA6W1/RA6W2 device.

4.1 Configure Application Parameters

Before building and flashing, update the WebSocket parameters in `websocket_client_app.c` to match your setup:

1. Server URI: Replace the IP address with that of the system running the WebSocket server.

```
#define WS_APP_SERVER_URI "wss://<ip_address>:8765"  /* Secure WebSocket (TLS) */
```

Figure 3. Updating the WebSocket server's IP address

2. Root CA Certificate: Replace the certificate with the contents of `cert.pem` generated from the OpenSSL commands provided earlier and format it accordingly.

```
#define WS_APP_ROOT_CA \
"-----BEGIN CERTIFICATE-----\n" \
"MIIBWzCCAQKgAwIBAgIUQai7wFktzIp869JxLI11+XwqV/4wCgYIKoZIzj0EAwIw\n" \
"FDESMBAGA1UEAwwJTX1NUVRUX0NBMB4XDTI2MDQyODEyNTE1NFoXDTM2MDQyNTEy\n" \
"NTE1NFowFDESMBAGA1UEAwwJTX1NUVRUX0NBMFkwEwYHKoZIzj0CAQYIKoZIzj0D\n" \
"AQcDQgAESqzC8ZkZHY1/wUeHCzLKnWF6iShILw1w05FTUHSDL5MFVaIIUa4Z0hE6\n" \
"pYiE4iyQBPRLRuCu68evbwLy/OWL+KMyMDAwHQYDVR0OBBYEFP8EjwaIzTWCzyae\n" \
"M1TRMsfq9w52MA8GA1UdEwEB/wQFMAMBAf8wCgYIKoZIzj0EAwIDRwAwRAIgdpqX\n" \
"2jN5djuohhPRYsYOhLApdJ+KN7KMJj2D7XzMUdsCIFHcmwMk3VSCcaXJWQmK00/X\n" \
"56Cb7r88xVaIATL/RUNC\n" \
"-----END CERTIFICATE-----\n"
```

Figure 4. Example of the CA certificate format

Generate project, build, and flash firmware using e^2 studio.

4.2 Program Firmware Image

The application is located in the directory: `application_projects/r19an0428/Websocket_Over_Mqtt/`. The firmware image is generated and flashed onto the EK-RA6W1/RA6W2 using e^2 studio with J-Link support.
Generate project content, build, and flash the firmware to the board using e^2 studio:

1. Open e^2 studio and connect the EK-RA6W1/RA6W2 board to the PC using USB (J-Link).
2. Import the project into e^2 studio.
3. Open `configuration.xml` and click **Generate Project Content**.
4. Build the project (**Project** > **Build Project**).
5. Go to **Debug Configurations** > **Startup** and select the firmware image file.
6. Click **Apply**, and then click **Debug**.

The firmware is automatically flashed to the device. After completion, the board reboots and begins execution.

4.3 Connect Device

1. When prompted by Easy Setup, configure the device in Station mode and connect to an AP. The AP must be the same one that the system running the server is connected to.

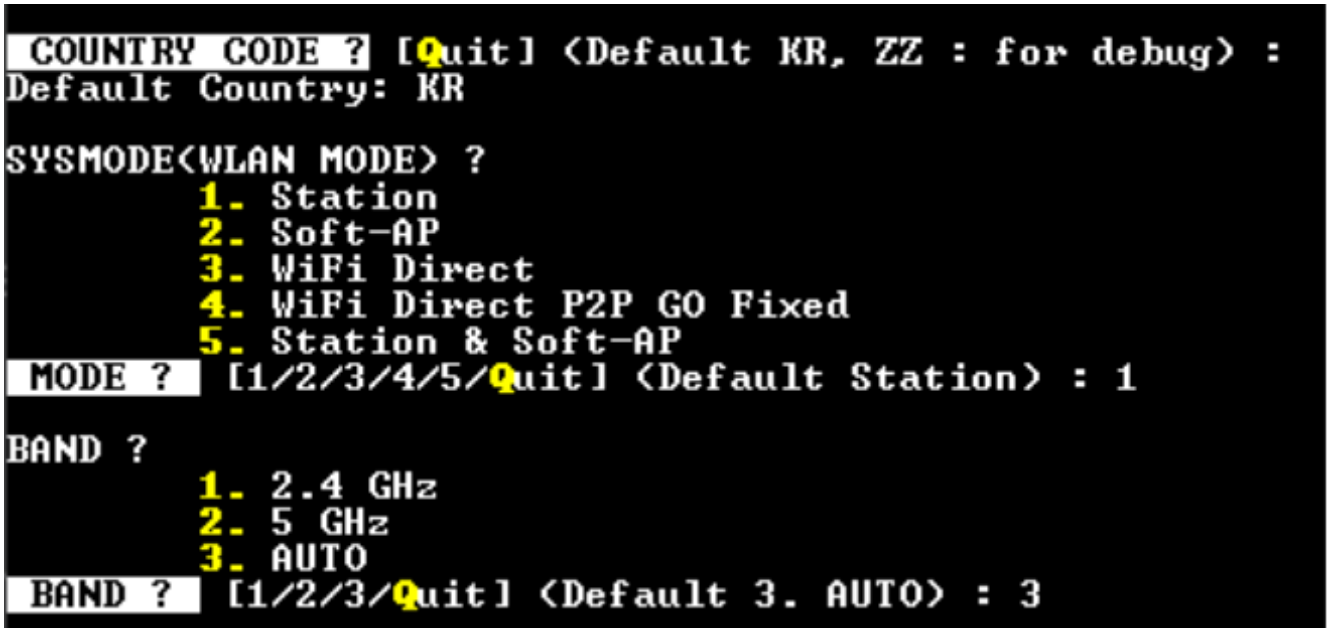


Figure 5. Setting up the device in Station mode

2. Enable the SNTP client when prompted. The default values are enough for this example to work:



Figure 6. Configuring the SNTP Client

3. The device connects to the AP first and then to the SNTP server:



Figure 7. Console log showing device connected to the AP and SNTP server

4. Connect to J-Link RTT viewer with the parameters shown in Figure 8. The address of the RTT control block can be found in the `.map` file under the `label .bss._SEGGER_RTT` generated in the `Debug` folder after the application is built in the e^2 studio IDE.

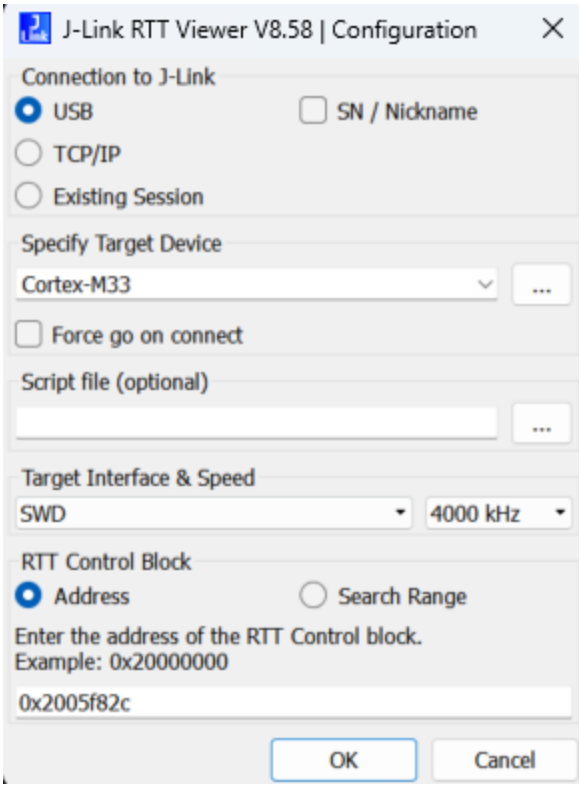


Figure 8. J-Link terminal connection parameters

5. When the SNTP connection is successful, the device attempts to establish a connection to the MQTT broker automatically. After this, the client sends MQTT CONNECT. The client then sends MQTT SUBSCRIBE to the topic `rrq61x_sub`. When subscribed, the application periodically publishes messages to the topic `rrq61x_pub`. This can be viewed on the J-Link terminal as well as the MQTT broker logs as seen in Figure 9 and Figure 10.

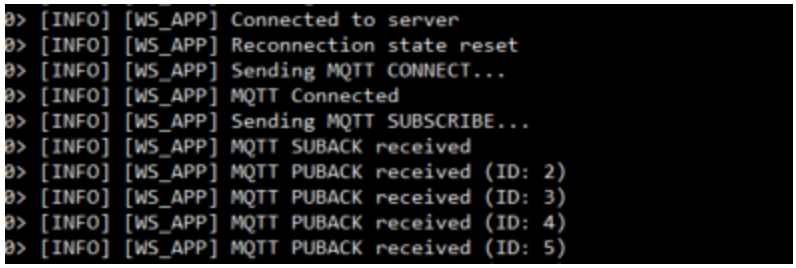


Figure 9. Receiving published messages on the J-Link terminal

```
1778157749: Sending CONNACK to RA6W1_WS (0, 0)
1778157754: Received SUBSCRIBE from RA6W1_WS
1778157754:     rrq61x_sub (QoS 1)
1778157754: RA6W1_WS 1 rrq61x_sub
1778157754: Sending SUBACK to RA6W1_WS
1778157759: Received PUBLISH from RA6W1_WS (d0, q1, r0, m2, 'rrq61x_pub', ... (15 bytes))
1778157759: Sending PUBLISH to auto-33BDA5D8-FDDB-7D37-BB20-72F343E5B0A9 (d0, q0, r0, m0, 'rrq6
1x_pub', ... (15 bytes))
1778157759: Sending PUBACK to RA6W1_WS (m2, rc0)
1778157764: Received PUBLISH from RA6W1_WS (d0, q1, r0, m3, 'rrq61x_pub', ... (15 bytes))
1778157764: Sending PUBLISH to auto-33BDA5D8-FDDB-7D37-BB20-72F343E5B0A9 (d0, q0, r0, m0, 'rrq6
1x_pub', ... (15 bytes))
1778157764: Sending PUBACK to RA6W1_WS (m3, rc0)
1778157769: Received PUBLISH from RA6W1_WS (d0, q1, r0, m4, 'rrq61x_pub', ... (15 bytes))
1778157769: Sending PUBLISH to auto-33BDA5D8-FDDB-7D37-BB20-72F343E5B0A9 (d0, q0, r0, m0, 'rrq6
```

Figure 10. Logs on the Mosquitto broker

6. Opening a terminal and publishing a message to the MQTT broker with the topic `rrq61x_sub` reflects in the J-Link logs of the device.

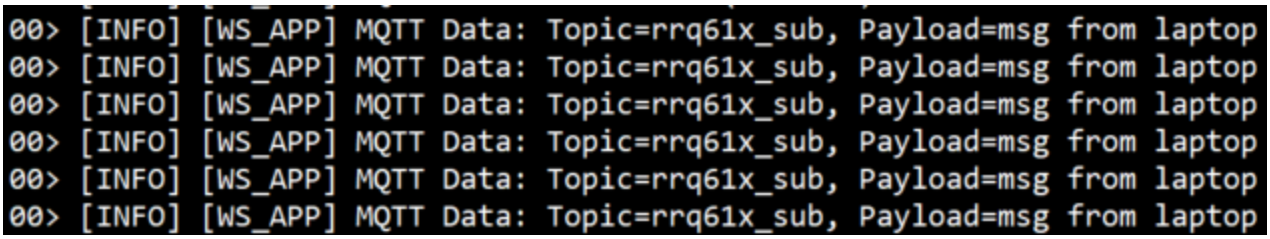


Figure 11. Message sent from PC received by the device

7. Similarly, the continuous publishing of the topic `rrq61x_pub` can be seen by subscribing to the broker.

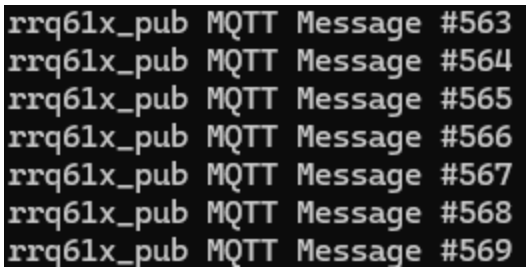


Figure 12. Logs on the Mosquitto subscriber terminal

4.4 Connection Management and MQTT Recovery

The application implements both WebSocket and MQTT reconnection behavior. The WebSocket reconnects automatically with exponential backoff whenever the underlying transport fails, while the MQTT state resets after disconnection and retries the CONNECT and SUBSCRIBE flow once the connection is restored. Pending QoS 1 packets are tracked, and timeouts are reported if a PUBACK is not received in time. Additionally, TCP keep-alive and WebSocket ping/pong mechanisms help preserve and maintain the transport connection.

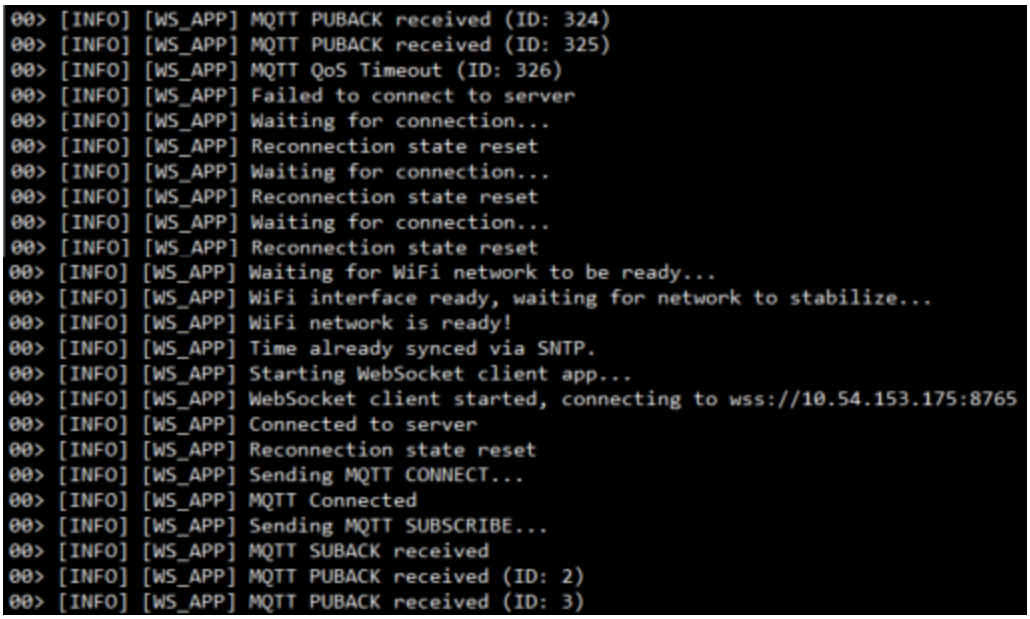


Figure 13. J-Link logs indicating disconnect and automatic reconnect

5. Revision History

Revision	Date	Description
1.02	Sep 21, 2026	Updated the Software requirements section.
1.01	July 21, 2026	Modified commands under Set Up MQTT Broker with WebSocket Support.
1.00	May 26, 2026	First version.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01)

CorporateHeadquarters

TOYOSU FORESIA, 3-2-24 Toyosu
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/

© 2026 Renesas Electronics Corporation. All rights reserved.