Web Socket Application

User guide

Version 0.2

May 2016

Redpine Signals, Inc.
2107 N. First Street, #540
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com

About this Document

This document describes the process of bringing up the RS9113 based module as a Websocket client.

Disclaimer:

Table of Contents

Table of Figures

Table of Tables

1 Introduction

This project is applicable to all the WiSeConnect variants like WiSeConnect Plus, WiSeMCU and WyzBee. The term WiSeConnect refers to its appropriate variant.

1.1 Websocket Overview

WebSocket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The WebSocket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. WebSocket enables streams of messages on top of TCP.

1.2 Application Overview

1.2.1 Overview

This application demonstrates how to configure device in client mode to open Web socket to transmit data over Websocket to Web Server.

1.2.2 Sequence of Events

This Application explains user how to:

- Connect the Device to an Access point and get IP address through DHCP
- Connect to Webserver opened on remote peer using websocket client
- Send data to websocket server

1.3 Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

1.3.1 SPI based Setup Requirements

- Windows PC with CooCox IDE
- Spansion (MB9BF568NBGL) micro controller

Note: If user does not have Spansion (MB9BF568NBGL) host platform, please go through the SPI-Porting guide **\sapis\docs\RS9113-WiSeConnect-SAPI-Porting-Guide-vx.x.pdf** for SAPIs porting to that particular platform.

- WiSeConnect device
- WiFi Access point
- Linux/Window PC with Websocket server and openSSL support (This application using no-poll server for webserver)

Note: Download No-Poll server from below link,

http://www.aspl.es/nopoll/downloads/nopoll-0.3.2.b232.tar.gz

1.3.2 UART/USB-CDC based Setup Requirements

- Windows PC with Dev-C++ IDE

- WiSeConnect device
- WiFi Access point
- Linux/Window PC with Websocket server and openSSL support (This application using no-poll server for webserver)

Note: Download No-Poll server from below link,

http://www.aspl.es/nopoll/downloads/nopoll-0.3.2.b232.tar.gz

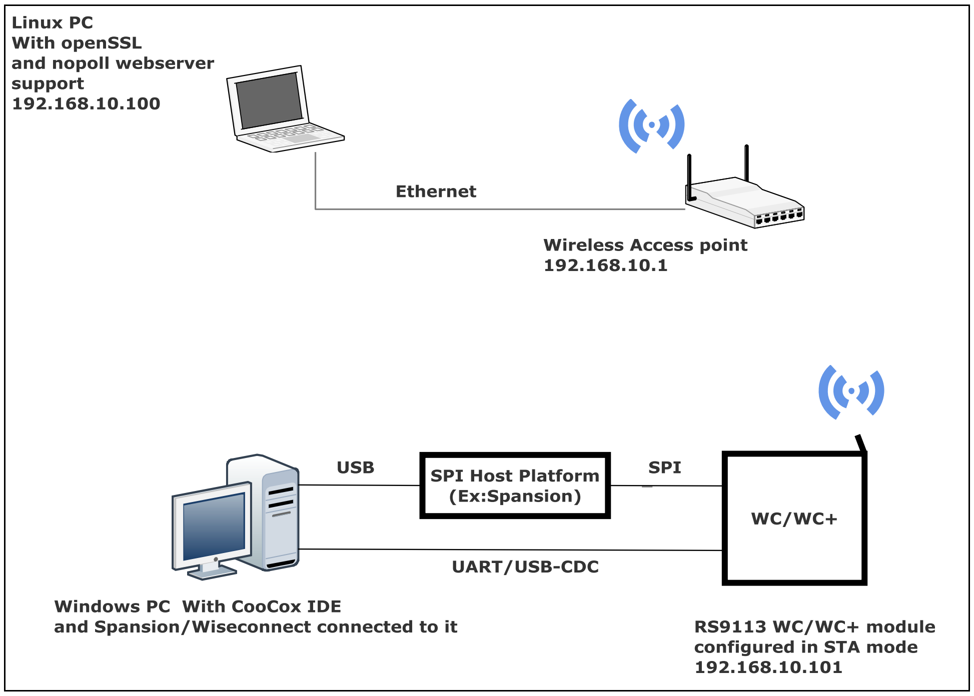


Figure 1: Setup Diagram

2 Configuration and Execution of the Application

The example application is available in the Release at **{Release $}/host/sapis/examples.** These examples will have to be initialized, configured and executed to test the application. The initialization varies based on the interface but configuration and execution are the common.

2.1 Initializing the Application

2.1.1 SPI Interface

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for opening the ***websocket_client*** example in CooCox IDE.

2.1.2 UART/USB-CDC Interface

If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for opening the ***websocket_client*** example in Dev-C++ IDE

2.2 Configuring the Application

1. Open ***sapis/examples/wlan/websocket_client/rsi_websocket_client_app.c*** file and update/modify following macros :

SSID refers to the name of the Access point.

```
#define SSID                        "<ap name>"
```

CHANNEL_NO refers to the channel in which device should scan. If it is 0, device will scan all channels.

```
#define CHANNEL_NO              0
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

Valid configuration is:

RSI_OPEN - For OPEN security mode

RSI_WPA - For WPA security mode

RSI_WPA2 - For WPA2 security mode

```
#define SECURITY_TYPE           RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
#define PSK                         "<psk>"
```

To Load certificate

```
#define  LOAD_CERTIFICATE           1
```

If **LOAD_CERTIFICATE** set to 1, application will load certificate which is included using rsi_wlan_set_certificate API.

By default, application loading "cacert.pem" certificate if **LOAD_CERTIFICATE** enable. In order to load different certificate, user has to follow the following steps :

– rsi_wlan_set_certificate API expects the certificate in the form of linear array. So, convert the pem certificate into linear array form using python script provided in the release package "sapis/examples/utilities/certificates/certificate_script.py"

  Ex: If the certificate is wifi-user.pem .Give the command in the following way

```
python certificate_script.py ca-cert.pem
```

  Script will generate wifiuser.pem in which one linear array named cacert contains the certificate.

– After conversion of certificate, update *rsi_ssl_client.c* source file by including the certificate file and by providing *the* required parameters to rsi_wlan_set_certificate API.

Note: Once certificate loads into the device, it will write into the device flash. So, user need not load certificate for every boot up unless certificate change.

So define `LOAD_CERTIFICATE` as 0, if certificate is already present iin the device.

Note: All the certificates are given in the release package.
Path: ***sapis/examples/utilities/certificates***

To Open Websocket client:

FLAGS refers to open normal websocket or wesocket over SSL with IPv4 or IPv6

If User wants to open normal websocket client set FLAGS to 0 or user wants to open Websocket over SSL then set FLAGS to 2 (WEBSOCKET_SSL). Default configuration is Normal Websocket client

```
        #define FLAGS                            0
```

Port number of the remote websocket server. Default configuration of server port number is Normal Websocket server.

```
#define SERVER_PORT                     1234
```

Note: If user wants to open Websocket over SSL then update SERVER_PORT macro with 1235 or 1236 as in nopoll SSL websocket server is running on port numbers 1235 and 1236

IP address of the remote websocket server

IP address should be configured in long format and in little endian byte order.

Example: To configure "192.168.10.101" as IP address, update the macro **SERVER_IP** as **0x650AA8C0**.

```
        #define SERVER_IP                        0x650AA8C0
```

Web socket resource name, maximum 50 characters

```
        #define WEB_SOCKET_RESOURCE_NAME     "<resource_name>"
```

Web socket host name, maximum 50 characters

```
      #define WEB_SOCKET_HOST_NAME          "<host name>"
```

Message to send remote server

```
      #define MESSAGE                       "<message>"
```

`FIN_BIT` is used to indicate whether it is the last packet or not.

In this example, **`FIN_BIT`** is setting in last packet of configured number of packets (**`NUMBER_OF_PACKETS`**). After receiving packet with FIN BIT set, websocket server sends back the received data to WiSeConnect device.

```
      #define FIN_BIT                       128
```

Number of packets to send

```
      #define NUMBER_OF_PACKETS 1000
```

Application memory length which is required by the driver

```
      #define  GLOBAL_BUFF_LEN       8000
```

To configure IP address

`DHCP_MODE` refers whether IP address configured through DHCP or STATIC

```
      #define DHCP_MODE   1
```

Note: If user wants to configure STA IP address through DHCP then set **`DHCP_MODE`** to 1 and skip configuring the following **`DEVICE_IP`**, **`GATEWAY`** and **`NETMASK`** macros.

(Or)

If user wants to configure STA IP address through STATIC then set **`DHCP_MODE`** macro to "0" and configure following **`DEVICE_IP`**, **`GATEWAY`** and **`NETMASK`** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure "192.168.10.10" as IP address, update the macro **`DEVICE_IP`** as **`0x0A0AA8C0`**.

```
      #define  DEVICE_IP           0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure "192.168.10.1" as Gateway, update the macro GATEWAY as **`0x010AA8C0`**

```
      #define  GATEWAY             0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure "255.255.255.0" as network mask, update the macro **`NETMASK`** as **`0x00FFFFFF`**

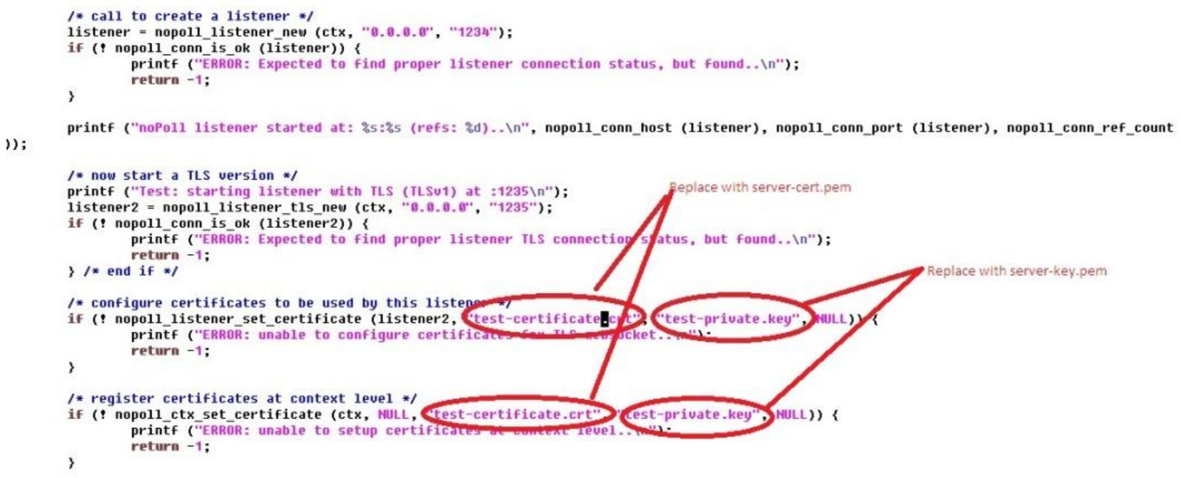
```
      #define   NETMASK            0x00FFFFFF
```

2. Open ***sapis/include/rsi_wlan_config.h*** file and update/modify following macros,

```
 #define  CONCURRENT_MODE             RSI_DISABLE
 #define  RSI_FEATURE_BIT_MAP         FEAT_SECURITY_OPEN
#define  RSI_TCP_IP_BYPASS           RSI_DISABLE
#define RSI_TCP_IP_FEATURE_BIT_MAP (TCP_IP_FEAT_DHCPV4_CLIENT
| TCP_IP_FEAT_SSL)
 #define  RSI_CUSTOM_FEATURE_BIT_MAP  0
 #define  RSI_BAND                    RSI_BAND_2P4GHZ
```

2.3 Executing the Application

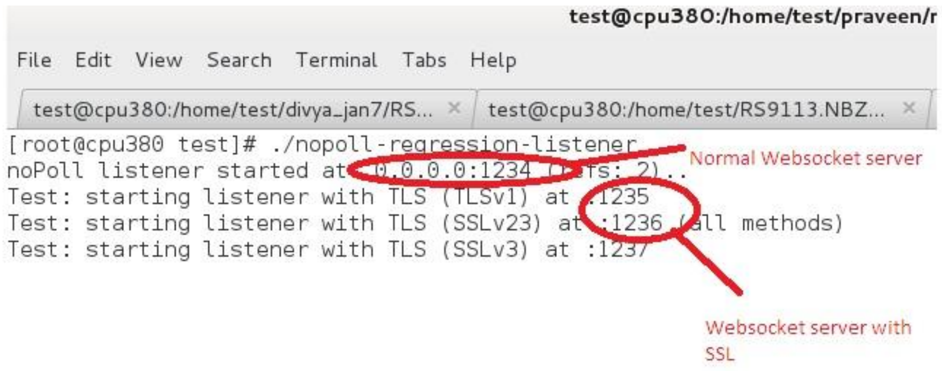
1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect WiSeConnect device in STA mode.
2. Install no-poll server in Linux/Windows PC and run Websocket server. Please follow the below steps to run no-poll server in LINUX PC,
   - Download No-Poll server from below link, **http://www.aspl.es/nopoll/downloads/nopoll-0.3.2.b232.tar.gz**
   - Unzip the package and do “. /configure”, “make” and “make install”**.**
   - After successful installation, go to test folder to run sample websocket server.
   - If user wants to test websocket over SSL, Copy server-cert.pem and server-key certificates from release package (sapis/examples/utilities/certificates) to test folder.
   - Open nopoll-regression-listener.c file and change certificates to server-cert.pem and server-key.pem in nopoll_listener_set_certificate(listener2, “server-cert.pem”,”server-key.pem”,NULL) and nopoll_ctx_set_certificate(ctx,NULL,”server-cert.pem”,”server-key.pem”,NULL) functions.



```
        /* call to create a listener */
        listener = nopoll_listener_new (ctx, "0.0.0.0", "1234");
        if (! nopoll_conn_is_ok (listener)) {
                printf ("ERROR: Expected to find proper listener connection status, but found..\n");
                return -1;
        }

        printf ("noPoll listener started at: %s:%s (refs: %d)..\n", nopoll_conn_host (listener), nopoll_conn_port (listener), nopoll_conn_ref_count
));

        /* now start a TLS version */
        printf ("Test: starting listener with TLS (TLSv1) at :1235\n");
        listener2 = nopoll_listener_tls_new (ctx, "0.0.0.0", "1235");
        if (! nopoll_conn_is_ok (listener2)) {
                printf ("ERROR: Expected to find proper listener TLS connection status, but found..\n");
                return -1;
        } /* end if */

        /* configure certificates to be used by this listener */
        if (! nopoll_listener_set_certificate (listener2, "test-certificate.crt", "test-private.key", NULL)) {
                printf ("ERROR: unable to configure certificates for TLS websocket..\n");
                return -1;
        }

        /* register certificates at context level */
        if (! nopoll_ctx_set_certificate (ctx, NULL, "test-certificate.crt", "test-private.key", NULL)) {
                printf ("ERROR: unable to setup certificates at context level..\n");
                return -1;
        }
```

   - Compile the source code by giving “make” command in test folder.

- Run server by giving “./nopoll-regression-listener” command.
- After running the nopoll server, it will open Four sockets on port numbers 1234, 1235, 1236 and 1237.
  1234 – For Normal Websocket server
  1235 – For TLSv1 websocket server
  1236 – For SSLv23 websocket server
  12367– For SSLv3 websocket server

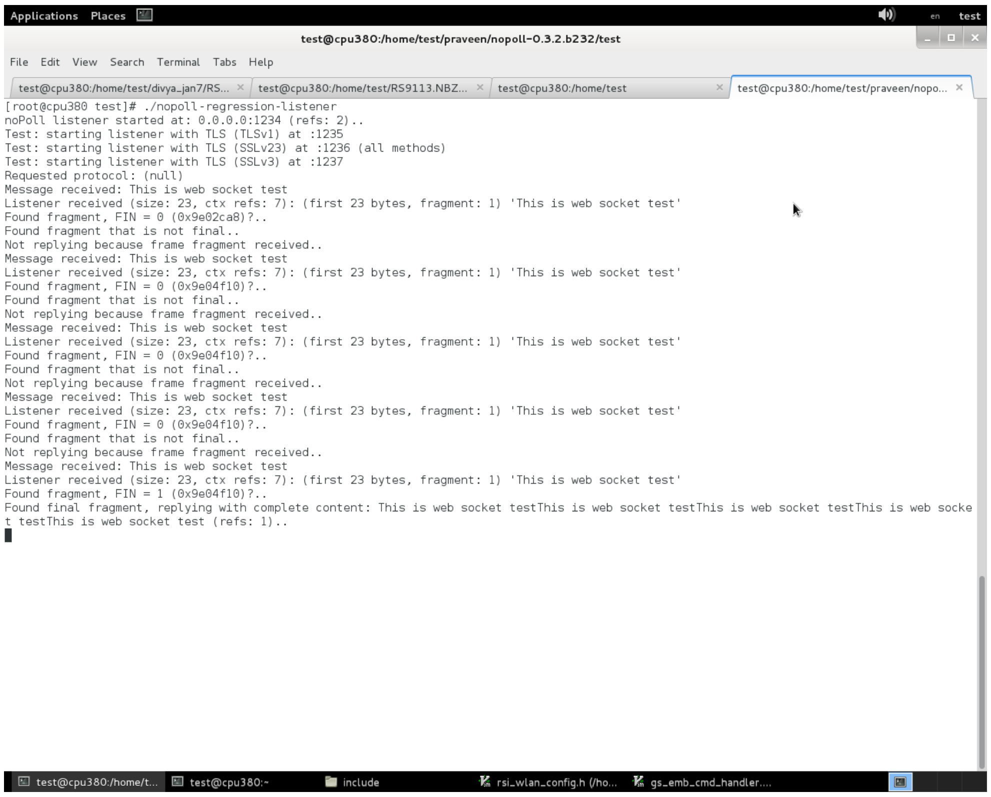

test@cpu380:/home/test/praveen/r
File Edit View Search Terminal Tabs Help
test@cpu380:/home/test/divya_jan7/RS...
test@cpu380:/home/test/RS9113.NBZ...
Normal Websocket server
Websocket server with SSL

```
[root@cpu380 test]# ./nopoll-regression-listener
noPoll listener started at: 0.0.0.0:1234 (refs: 2)..
Test: starting listener with TLS (TLSv1) at :1235
Test: starting listener with TLS (SSLv23) at :1236 (all methods)
Test: starting listener with TLS (SSLv3) at :1237
```

3. **SPI Interface**
   If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for executing the ***websocket_client*** example in CooCox IDE.
4. **UART/USB-CDC Interface**
   If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for executing the ***websocket_client*** example in Dev-C++ IDE

5. After the program gets executed, WiSeConnect Device would be connected to Access point having the configuration same as that of in the application and get IP.
6. The Device which is configured as Websocket client will connect to remote SSL server and sends number of packets configured in **`NUMBER_OF_PACKETS`**.

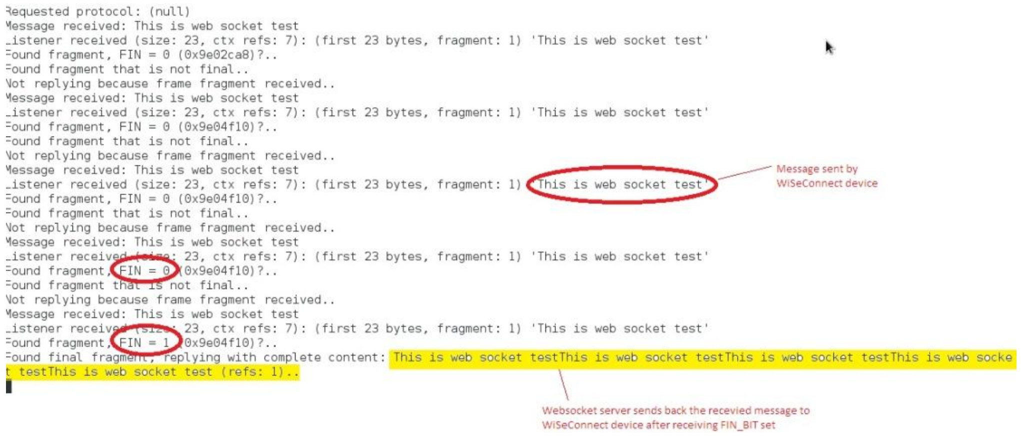
Please refer the given below image for Data Rx on Websocket server.



Applications Places
test@cpu380:/home/test/praveen/nopoll-0.3.2.b232/test
File Edit View Search Terminal Tabs Help
test@cpu380:/home/test/divya_jan7/RS...
test@cpu380:/home/test/RS9113.NBZ...
test@cpu380:/home/test
test@cpu380:/home/test/praveen/nopo...
test@cpu380:/home/t...
test@cpu380:~
include
rsi_wlan_config.h (/ho...
gs_emb_cmd_handler....

```
[root@cpu380 test]# ./nopoll-regression-listener
noPoll listener started at: 0.0.0.0:1234 (refs: 2)..
Test: starting listener with TLS (TLSv1) at :1235
Test: starting listener with TLS (SSLv23) at :1236 (all methods)
Test: starting listener with TLS (SSLv3) at :1237
Requested protocol: (null)
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e02ca8)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 1 (0x9e04f10)?..
Found final fragment, replying with complete content: This is web socket testThis is web socket testThis is web socket testThis is web socke
t testThis is web socket test (refs: 1)..
```

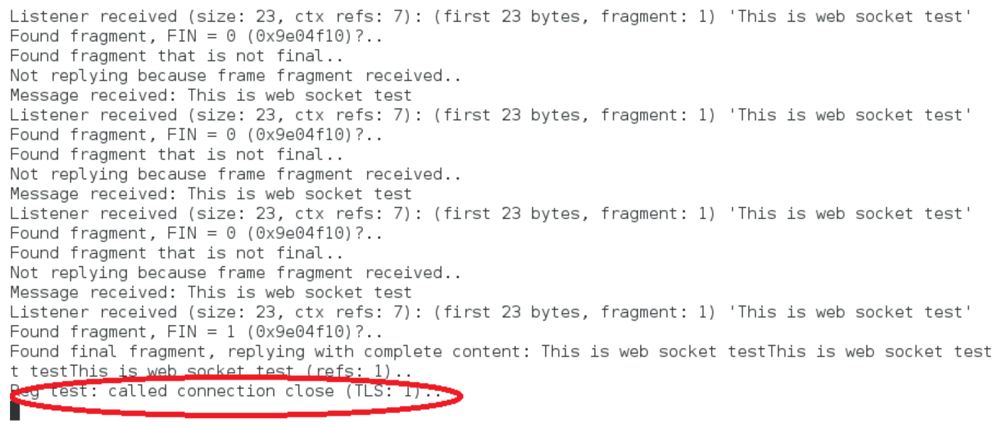
7. From application, setting FIN_BIT in last data packet. After receiving data packet with FIN_BIT set, Websocket server sends back the data received until FIN_BIT set to WiSeConnect device. Please find below image for websocket server sends back data to WiSeConnect device after receiving packet with FIN_BIT set,



```
Requested protocol: (null)
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e02ca8)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 1 (0x9e04f10)?..
Found final fragment, replying with complete content: This is web socket testThis is web socket testThis is web socket testThis is web socke
t testThis is web socket test (refs: 1)..
```

Message sent by WiSeConnect device
Websocket server sends back the recevied message to WiSeConnect device after receiving FIN_BIT set

8. WiSeConnect device will receive the message sent by Websocket server and initiate connection close to websocket server. Please refer the given below image for connection close at websocket server.



```
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 0 (0x9e04f10)?..
Found fragment that is not final..
Not replying because frame fragment received..
Message received: This is web socket test
Listener received (size: 23, ctx refs: 7): (first 23 bytes, fragment: 1) 'This is web socket test'
Found fragment, FIN = 1 (0x9e04f10)?..
Found final fragment, replying with complete content: This is web socket testThis is web socket test
t testThis is web socket test (refs: 1)..
Reg test: called connection close (TLS: 1)..
```