Airoha IoT SDK for RTOS Internet and Open Source Software Guide

Version: 3.5

Release date: 15 September 2017

Document Revision History

Revision	Date	Description
1.0	7 March 2016	Initial release
2.0	2 May 2016	Added the FatFs module.
3.0	30 June 2016	Added CMSIS and LZMA.
3.1	2 September 2016	Added the illustration of FreeRTOS memory configuration and settings.
3.2	13 January 2017	Added the mDNS and WebSocket.
3.3	5 May 2017	Updated DHCPD
3.4	1 August 2017	Updated license information
3.5	15 September 2017	Added AWS IoT SDK support

Table of Contents

Lists of tables and figures

1. Overview

In the new IoT era, devices communicate with various Internet of Things (IoT) protocols to form a smart network. Many new protocols are recently defined to meet the requirements of machine-to-machine communication on devices with constrained resources. Open standards enable the IoT growth in the near future.

Airoha offers IoT SDK v4 with mainstream open source software to help developers build their projects with globally available resources and a full control over the software. The adopted standards of IoT are still changing, the SDK enables developers to follow the updates of new specifications rapidly.

The SDK includes all major protocol support, such as the fundamental TCP/IP stack, the TLS library for secure data transport, and many common application protocols developed recently for the new machine-to-machine communication, including HTTP, SNTP, and DHCP daemon (DHCPD). Besides the communication protocols, the SDK also includes XML and JSON libraries for packaging and parsing data. When building these protocols, memory footprint is a major requirement. Airoha provides competitive packages with small memory footprint and essential functions. These software packages are used to build applications and connect devices easily with the IoT world.

The SDK requires RTOS as an underlying OS, more specifically FreeRTOS; the leading real-time operating system in the embedded systems market. All the open source software is distributed in source release, therefore, the developers can make any changes to fit their requirements and redistribute the software under the rights permitted by the corresponding license.

This guide briefs the open source software bundled in the SDK. It provides information about the open source software packages and guides the developers to design, prototype and implement IoT projects in a convenient environment.

As an open source software package, the information is already available in various sources. This guide is an easier referencing to the module descriptions including the official web site and the hardware or software integration versions. Developers can access the latest source code from the official website and merge or replace the one in the package with a corresponding version. From the footprint statistics result, developers can have an estimate about the code size to set up the flash memory layout. Before commencing your own IoT project or application development, look for example applications and their source codes. Finally, the troubleshooting and limitations chapter provides more detailed information on the reported issues for the application development.

1.1. Open source software architecture of the SDK

The SDK integrates several widely adopted protocol implementations to reduce the development effort and enable device communication over the IP channel. The open source communication protocols and libraries included in the SDK are; HTTP, JSON, XML, SNTP, SSL/TLS, DHCPD, lwIP (IPv4) Lempel–Ziv–Markov chain algorithm (LZMA) decoder and WebSocket. They provide a communication layer between applications and the FreeRTOS. The architecture layout of the SDK is shown in Figure 1. lwIP is a TCP/IP open source protocol that provides transport for sending and receiving data between the nodes connected through the Internet. The HTTP client module in the SDK is an implementation of client protocol to GET or POST data to the remote HTTP server. SSL/TLS establishes secure data transport protection on TCP connections. Running HTTP over SSL/TLS enables secure data transfer. SNTP is a protocol to obtain the current time in high precision from the server. DHCPD is used in soft access point (AP) mode to handle IP address management to the connected Wi-Fi station nodes. JSON and XML are libraries to parse and generate the data in a specific format. LZMA decoder decodes the compressed data encoded by the LZMA encoder. The WebSocket provides full-duplex communication channels over TCP, through which the server is able to send content without being solicited by the client.

Four components, MQTT (Message Queuing Telemetry Transport) client, HTTP server, HTTP/2 client and mDNS server are added into the SDK. MQTT is a machine-to-machine (M2M)/IoT connectivity protocol. It was designed as an extremely lightweight publish or subscribe messaging transport. HTTP server enables developers to host web

services to provide information for clients or perform operations with Common Gateway Interface (CGI) by invoking the associated programs. HTTP/2 is the successor of HTTP/1.1 developed based on Google's SPDY, and finalized in RFC 7540. HTTP/2 is designed to make the most of the available bandwidth through compression and concurrent downloads, and with the ability for servers to push content to users makes it more convenient to preload and deliver data in real-time. Developers can use nghttp2, an implementation of HTTP/2 and its header compression algorithm HPACK in C in the SDK, to implement client applications to communicate with HTTP/2 servers. The mDNSResponder project is a component of Apple's Bonjour. Developers can use mDNSResponder to publish services as Bonjour server application.

AWS IoT provides secure, bi-directional communication between Internet-connected things, including sensors, actuators, embedded devices or smart appliances, and the AWS cloud. This enables collecting telemetry data from multiple devices then storing and analyzing the data. It also enables to create applications that enable users to control these devices from their phones or tablets.

In addition, the open source FatFs is included to provide the FAT (File Allocation Table) based file and directory operations. The Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals.

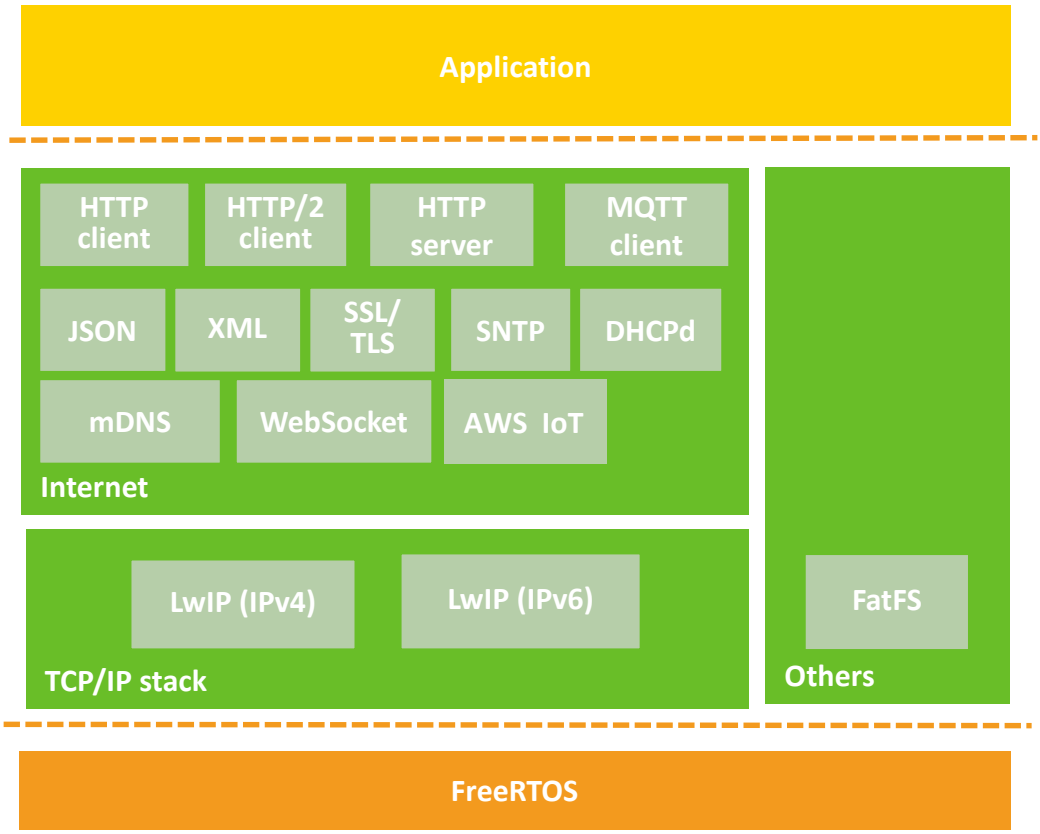

Figure 1. Architecture layout of the SDK's open source software

1.2. Open source software resources for the SDK

The open source software packages in the SDK are listed in Table 1 and for developer's reference only. Developer acknowledges that such listed open source software may be supplemented or amended by Airoha from time to time. Developer must comply with all licensing terms applicable to such open source software. Airoha makes the following disclaimers regarding the open source software on behalf of itself, and the copyright holders, contributors, and licensors of the listed open source software: TO THE FULLEST EXTENT PERMITTED UNDER APPLICABLE LAW, THE OPEN SOURCE SOFTWARE ARE PROVIDED BY THE COPYRIGHT HOLDERS, CONTRIBUTORS,

LICENSORS, AND AIROHA "AS IS" AND ANY REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER ORAL OR WRITTEN, WHETHER EXPRESS, IMPLIED, OR ARISING BY STATUTE, CUSTOM, COURSE OF DEALING, OR TRADE USAGE, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, ARE DISCLAIMED. IN NO EVENT WILL THE COPYRIGHT OWNER, CONTRIBUTORS, LICENSORS, OR AIROHA BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THE OPEN SOURCE SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Table 1. Open source packages

Module	Open source software licenses	Comments
RTOS	FreeRTOS License: Modified GPL Please refer to license disclaimer in kernel/rtos/FreeRTOS/Source/include/FreeRTOS.h Note: FreeRTOS V10 and later version are distributed under the MIT license. http://www.freertos.org/a00114.html	Market leading, de-facto standard OS for embedded systems.
TCP/IP	lwIP License: BSD License, MediaTek EULA http://lwip.wikia.com/wiki/License	lwIP (lightweight IP) is a widely used open source TCP/IP stack designed for embedded systems. Several files are modified by Airoha for IPv6.
SSL/TLS	mbed TLS License: Apache 2.0 License http://www.apache.org/licenses/LICENSE-2.0	Easy to use SSL/TLS library including cryptographic and SSL/TLS capabilities with small footprint.
HTTP (1.1) client	mbed HTTP client License: MIT License, Airoha License https://developer.mbed.org/users/WiredHome/code/HTTPClient/file/3556275bebf3/HTTPClient.cpp	A small HTTP client supporting HTTP and HTTPS. The porting from `.cpp` to `.c` was implemented by Airoha.
XML	Mini-XML License: LGPLv2 with static linking exception http://michaelrsweet.github.io/mxml/	A small XML library to read and write XML documents.
JSON	cJSON License: MIT License https://github.com/kbranigan/cJSON/blob/master/LICENSE	An ultra-lightweight, portable single-file, simple-as-can-be ANSI-C compliant JSON parser.
SNTP	lwIP-contrib SNTP License: Modified BSD License http://lwip.wikia.com/wiki/License	Implementation of an SNTP client to retrieve the GMT time from servers.
MQTT client	Eclipse Paho Embedded MQTT C/C++ client License: (Dual license: EPL or EDL) http://www.eclipse.org/org/documents/epl-v10.php	The Paho project provides an open-source client implementation of the MQTT messaging protocols.

Module	Open source software licenses	Comments
HTTP server	axTLS License: BSD License http://choosealicense.com/licenses/bsd-3-clause/	A small HTTP server to handle HTTP client requests and send a response.
HTTP/2 client	nghttp2 License: MIT License https://github.com/tatsuhiro-t/nghttp2	nghttp2 is an implementation of HTTP/2 and its header compression algorithm HPACK in C.
FatFs	FatFs License: BSD-style licenses http://elm-chan.org/fsw/ff/pf/appnote.html	FatFs is a generic FAT file system module for small embedded systems.
CMSIS	CMSIS License: BSD License https://developer.mbed.org/blog/entry/CMSIS-Components-BSD-Licensed/	CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces.
LZMA decoder	LZMA License: public domain http://www.7-zip.org/sdk.html	LZMA decoder is extracted from the LZMA SDK for further use.
mDNS server	mDNSResponder License: Apache 2.0 License https://opensource.apple.com/tarballs/mDNSResponder/	mDNSResponder is an implementation of mDNS and DNS-SD protocols.
WebSocket	Librws License: MIT License https://github.com/OlehKulykov/librws	Librws is a tiny and cross platform WebSocket client C library.
AWS IoT	`aws-iot-device-sdk-embedded-C` License: Apache License 2.0 https://github.com/aws/aws-iot-device-sdk-embedded-C	The AWS IoT device SDK for embedded C is a collection of C source files that can be used in embedded applications to securely connect to the AWS IoT platform.

The SDK open source packages are implemented by the active open source community with widely available online resources and forum support. The list of online resources is provided for each module in Table 2. The integrated versions are also included for developers to merge hot bug fixes or new features directly from the official release links.

Table 2. Online resources for each module in the SDK

Module	Official website	Integrated version	Online API reference
RTOS	http://www.freertos.org/RTOS.html	8.2.0	http://www.freertos.org/a00106.html
TCP/IP	http://savannah.nongnu.org/projects/lwip/ http://lwip.wikia.com/wiki/LwIP_Wiki	git://git.sv.gnu.org/lwip.git ssh://git.sv.gnu.org/srv/git/lwip.git http://git.savannah.gnu.org/r/lwip.git Commit ID: e448a9a	http://static2.wikia.nocookie.net/__cb20100724070440/mini6/images/0/0e/Lwip.pdf

Module	Official website	Integrated version	Online API reference
SSL/TLS	https://tls.mbed.org/	2.1.0	https://tls.mbed.org/api/
HTTP (1.1) client	https://developer.mbed.org/users/WiredHome/code/HTTPClient	Revision: 34:3556275bebf3	No online API. Exported API with comments can be found at `<sdk_root>/middleware/third_party/httpclient/inc/httpclient.h`.
XML	http://www.msweet.org/projects.php?Z3	Mini-XML 2.9	http://www.msweet.org/documentation/project3/Mini-XML.html
JSON	https://github.com/kbranigan/cJSON	https://github.com/kbranigan/cJSON/commits/master Commit ID: 9b463a0	No online API. Please find exported API at `<sdk_root>/middleware/third_party/cjson/inc/cJSON.h`.
SNTP	http://savannah.nongnu.org/projects/lwip/ http://lwip.wikia.com/wiki/LwIP_Wiki	Commit ID: 06dee8d	No online API. Please find exported API at `<sdk_root>/middleware/third_party/sntp/inc/sntp.h`.
DHCPD	The source code is Airoha proprietary implementation. No official website.	N/A.	No online API. Exported API with descriptions can be found at `<sdk_root>/middleware/MTK/dhcpd/inc/dhcpd.h`.
MQTT client	https://www.eclipse.org/paho/clients/c/embedded/	1.0.0	https://www.eclipse.org/paho/clients/c/embedded/
HTTP server	http://axtls.sourceforge.net/	1.5.3	No online API. Exported API with comments can be found at `<sdk_root>/middleware/third_party/httpd/inc/httpd.h`
HTTP/2 client	https://nghttp2.org/	https://github.com/tatsuhiro-t/nghttp2 Commit ID: 7d481db	https://nghttp2.org/documentation/apiref.html
FatFs	http://elm-chan.org/fsw/ff/00index_e.html	R0.12(1)	http://elm-chan.org/fsw/ff/00index_e.html
CMSIS	http://www.keil.com/pack/doc/CMSIS/General/html/index.html	4.00	http://www.keil.com/pack/doc/CMSIS/Core/html/modules.html
LZMA decoder	http://www.7-zip.org/sdk.html	15.08	http://www.7-zip.org/sdk.html
mDNS server	https://developer.apple.com/bonjour/	544	https://developer.apple.com/bonjour/

Module	Official website	Integrated version	Online API reference
WebSocket	https://github.com/OlehKulykov/librws	1.2.3	https://github.com/OlehKulykov/librws
AWS IoT	https://github.com/aws/aws-iot-device-sdk-embedded-C	2.1.1	https://github.com/aws/aws-iot-device-sdk-embedded-C

(1) FatFs version is R0.12 after SDK version 4.2.0 and R0.11 before SDK version 4.2.0.

The source location for the modules listed in Table 2 is shown in Figure 2. The FreeRTOS for this SDK release is under the `kernel` directory and the internet protocol middleware can be found under the `middleware` directory.

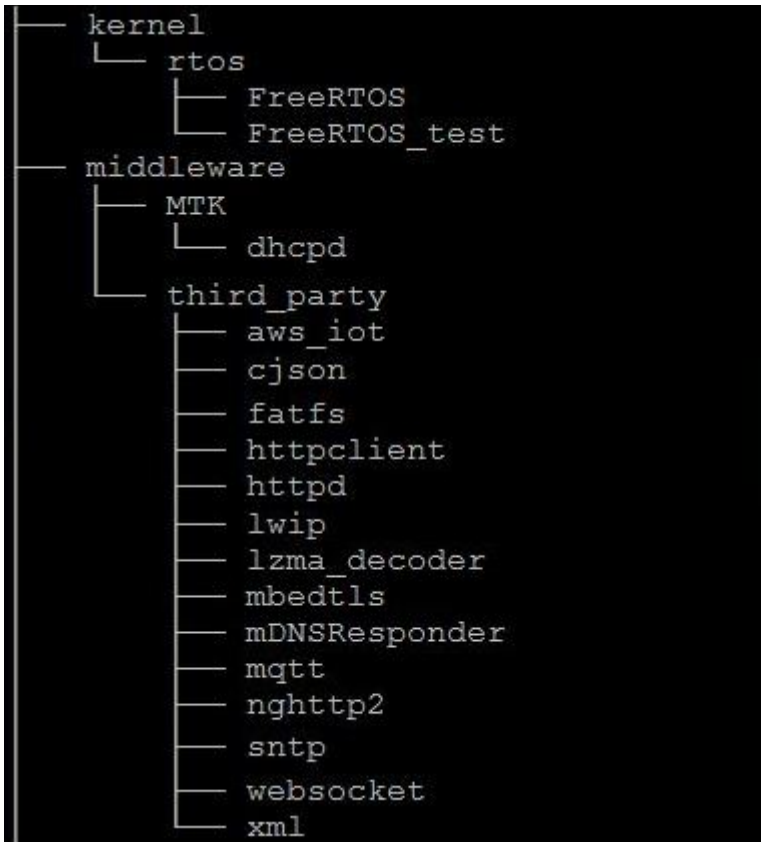

Figure 2. Source location of SDK open source software

Each module will be given a more detailed description in the upcoming sections.

2. RTOS: FreeRTOS

FreeRTOS is a real-time OS that manages a multitasking and multiprocessing system environment. It has a scheduler to manage user created tasks. FreeRTOS provides APIs for users to control, synchronize and communicate among various tasks.

2.1. Features

FreeRTOS supports the following five features to accomplish event/task scheduling and multitasking:

- Task and Scheduler. Each application consists of tasks or threads controlled by the operating system. The multitasking operation is implemented with a scheduler. The scheduler is in the kernel and manages the task execution at a specific time. The kernel can suspend and resume a task many times during lifecycle of the task execution.
- Queue. Queues are the primary forms of inter-task communications. In most cases they are used as thread safe first-in-first-out (FIFO) buffers.
- Semaphore. Threads use semaphores to control the access to shared resources.
- Software Timer. A software timer allows a function to be invoked at a predefined time. The timer callback functions are executed within the timer service task. It is essential to ensure the timer callback functions perform lightweight operations and return as quick as possible to avoid blocking the system resources.
- Event Group (enabled after FreeRTOS version 8.0.0). An event group is a set of event bits. Individual event bits within an event group are referenced by a bit number. Event bits are used to indicate if an event has already occurred or not. Event groups can also be used to synchronize tasks.
- For more information on FreeRTOS and its features, please refer to the official website.

2.2. Memory usage

Brief details on the memory usage can be found in FreeRTOS FAQ -
Memory Usage, Boot Times & Context Switch Times website, under sections.

- How much RAM does FreeRTOS use?
- How much ROM/Flash does FreeRTOS use?

2.3. Heap service

`heap_4.c` is used as the heap service on the SDK. The implementation is extended with more algorithms that are not supported by the official FreeRTOS. More details can be found in the header file `\gva\kernel\rtos\FreeRTOS\Source\include\portable.h`.

Few APIs are described in Table 3.

Table 3. Heap service APIs

Prototype	Description
`void *pvPortCalloc(size_t nmemb, size_t size)`	Provides the same functionality as the ISO C function `calloc()`. Allocates memory for an array "`nmemb`" with elements in bytes and returns a pointer to the allocated memory.

Prototype	Description
	The memory is set to zero. This function is available in SDK v1 and later.
`void *pvPortRealloc(void *pv, size_t size)`	Provides the same functionality as the ISO C function `realloc()`. Changes the size of the memory block pointed by "`pv`" to "`size`" in bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes. If the new size is larger than the old size, the added memory will not be initialized. This function is available in SDK v1 and later.
`void *pvPortMallocNC(size_t xWantedSize)`	This function is similar to `pvPortMalloc()`. Allocates "`xWantedSize`" bytes and returns a pointer to the allocated memory, except the allocated memory is in non-cacheable region. This function is available in SDK v3 and later.
`void vPortFreeNC(void *pv)`	This function is similar to `vPortFree()`. Frees the memory space pointed by "`pv`". All cache related operations are directly implemented in the API and is transparent to users. This function is available in SDK v3 and later.

For performance critical operations, such as task stack, heap is in cacheable region by default. The functions `pvPortMallocNC()` and `vPortFreeNC()` are implemented for temporary non-cacheable memory requirements, such as when using DMAs.

For convenient migration of third-party software and libraries, the SDK also supports C standard library heap implementation, including `malloc()`, `calloc()`, `realloc()` and `free()`. However, these functions are wrapped in FreeRTOS heap service functions `pvPortMalloc()`, `pvPortCalloc()`, `pvPortRealloc()` and `vPortFree()`, and require building the project with FreeRTOS module.

2.4. Examples

The source code and API documentation of the FreeRTOS can be found here.

2.5. Customization

Users can provide custom configuration for the FreeRTOS in `FreeRTOSConfig.h` header file. The configurable parameters include OS tick frequency, maximum priorities, disabled functions, etc.

For more details, please refer to the online documentation.

2.6. Limitations

Airoha does not impose any limitation to the integrated FreeRTOS. Please refer to the official online resources for more detailed information.

2.7. Developer notes

- There is no software restriction on the number of tasks to create.
- The tasks can share the same priority.
- If the configuration `USE_PORT_OPTIMISED_TASK_SELECTION` is enabled, the maximum number of task priorities will be 32 (0 to 31) in the ARM Cortex-M4 with floating point porting.
- Only API functions that end in "`FromISR`" can be called from within an interrupt service routine. Please refer to Open RTOS API documentation for more details.
- APIs that can potentially cause a context switch must not be called while the scheduler is suspended.
- APIs that can potentially cause a context switch must not be called from within a critical section.

3. TCP/IP: lwIP

TCP/IP (Transmission Control Protocol/Internet Protocol) is a communication internet protocol and can also be used in a private network, either an intranet or an extranet. It provides specifications on how data should be packetized, addressed, transmitted, routed and received at the destination. The current IoT standard is moving towards IP communication and transporting data over various physical layers such as Wi-Fi, IEEE 802.15.4 and Bluetooth.

3.1. Features

lwIP is a widely used open source TCP/IP stack designed for embedded systems. It includes the IP, ICMP, TCP, UDP, IGMP, ARP, AutoIP, DHCP, DNS and SNMP protocols. The SDK provides the following supported features for these protocols.

- IPv4 (LWIP_IPV4).
- UDP (LWIP_UDP): User Datagram Protocol, the widely adopted connectionless transmission protocol.
- TCP (LWIP_TCP): Transmission Control Protocol, a widely used transport protocol providing reliable and in-order delivery.
- ARP (LWIP_ARP).
- ICMP (LWIP_ICMP).
- DHCP (LWIP_DHCP).
- DNS (LWIP_DNS).
- NETCONN (LWIP_NETCONN).
- Socket (LWIP_SOCKET).

3.2. Memory usage

The MEM_SIZE parameter defines the size of the heap memory, PBUF_RAM, stores the sent and received data. If the application requires more data to send, the value of the parameter must be set higher.

The values of different control blocks are configurable in lwIP. For example, MEMP_NUM_NETDB sets the concurrent Domain Name Resolution connections. Table 4 lists the current configuration values of these blocks in the SDK. These pools are configured and allocated from a buffer reserved only for the lwIP. The values are set to pass internal performance test and are expected to fulfill most common use cases. However, the number of configured control blocks could limit the maximum concurrent connections created by the network applications in the system, developers can configure these values for a specific use case.

Table 4. Number of control blocks and buffers in lwIP

Name	Current value
MEMP_NUM_UDP_PCB	4
MEMP_NUM_TCP_PCB	8
MEMP_NUM_TCP_PCB_LISTEN	16
MEMP_NUM_TCP_SEB	255
MEMP_NUM_REASSDATA	5
MEMP_NUM_NETBUF	2

Name	Current value
MEMP_NUM_NETCONN	5
MEMP_NUM_TCPIP_MSG_API	8
MEMP_NUM_TCPIP_MSG_INPKT	8
MEMP_NUM_SYS_TIMEOUT	16
MEMP_NUM_NETDB	4
MEMP_NUM_PBUF	8
PBUF_POOL_SIZE	8

The required code size of lwIP is listed in Table 5. This information is gathered from an ARM Cortex M4 targeted configuration using the `gcc -Os` optimization. The feature set is IPv4, TCP, UDP, DHCP client, ICMP, RAW, NETCONN and Sockets and DNS client. The footprint is shown below.

Table 5. Footprint of the lwIP

Static footprint	ROM (bytes)	RAM (bytes)
Debug release	68121	59466

Note: If the options `LWIP_DEBUG`, `LWIP_ERROR`, `LWIP_ASSERTS` or `LWIP_STATS` are enabled, then the application has a significantly larger code footprint. Similarly, if the application is built with the compiler `-O0` optimization flag on, the footprint is again significantly affected.

3.3. Examples

1) File: `<sdk_root>/project/mt7687_hdk/apps/lwip_socket/src/main.c.`
2) Application Name: UDP Client/Server test + TCP Client/Server test.
3) Application Overview: This is a reference application to demonstrate the usage of socket functions. The UDP can be used as a client or a server. When operating as a client, it can send a data packet to a remote UDP server. Whereas, when it's a server, it can receive data packets from remote UDP clients.
4) Similar test can be carried out on TCP. When TCP operates as a client, it can connect to and communicate with a remote TCP server. When it operates as a server, it listens to and waits for incoming connections from remote TCP clients. After connection is established, it can communicate with the peer clients.

3.4. Customization

The custom settings and configuration can be applied in `otp.h` file located under `<sdk_root>/middleware/third_party/lwip/src/include/lwip/`. This file is fully commented and it's clear which of the options are defined, enabled or disabled.

Developers can also customize the project settings in `lwipopts.h` file located under `<sdk_root>/project/mt7687_hdk/apps/<project>/include/lwipopts.h`. For more information, please refer to introduction to lwIP in Wikia.

To enable or disable a feature, simply change configuration parameters in `lwipopts.h`. For example, if you would like to disable DNS and enable DHCP, please modify or add the following lines in `lwipopts.h` header file.

- //Disable DNS

```
#define LWIP_DNS 0
```

- //Enable DHCP

```
#define LWIP_DHCP 1
```

Here are the major features in lwIP that can be customized (enabled or disabled) by developers.

- LWIP_IPV4: Enables IPv4 protocol.
- LWIP_IPV6: Enables IPv6 protocol. In this release, the SDK supports only IPv4. Developers can manually enable it. The basic functions work with IPv4/IPv6 dual stack support, but it cannot pass the IPv6 conformance test.
- LWIP_UDP: Enables the UDP.
- LWIP_TCP: Enables the TCP.
- LWIP_RAW: The raw API is an event-driven API designed to be used without an operating system that implements zero-copy send and receive operation. However, the SDK suggests using BSD socket APIs (defined by LWIP_SOCKET) to make the socket application portable. If you are interested in the raw API, please refer to lwip native API documentation for more details.
- LWIP_ARP: Enables the ARP functionality.
- LWIP_ICMP: Enables the ICMP module in the IP stack.
- LWIP_IGMP: Enables the IGMP module (multicast support).
- LWIP_SNMP: Enables the lwIP SNMP agent. The UDP must be available for the SNMP transport.
- LWIP_DHCP: Enables the DHCP client module.
- LWIP_AUTOIP: Enables the AUTOIP module and the RFC 3927 - Dynamic Configuration of IPv4 Link-Local Addresses.
- LWIP_DNS: Enables the DNS module and requires UDP for DNS transport.
- LWIP_NETCONN: Enables Netconn API, requires using `api_lib.c` source file.
- LWIP_SOCKET: Enables Socket API, require using `sockets.c` to include a set of APIs compatible with POSIX-/BSD sockets.
- LWIP_STATS: Enables statistics collection in `lwip_stats`.
- LWIP_DEBUG: Enables debugging.
- LWIP_ERROR: Enables error logging.
- LWIP_NOASSERT: Disables LWIP_ASSERT checks.

3.5. Limitations

The list of feature limitations for lwIP is provided below.

- Doesn't support Selective Acknowledgements (SACKs) from RFC2018 in the TCP implementation.
- Only supports a few ICMP packet types, such as echo reply, destination unreachable and time exceeded. Most of the other types that are not specific to the embedded systems are ignored.

- Doesn’t support Network Address Translation (NAT) to map IP address space into another address for forwarding the IP packets.
- Few of TCP/IP options are supported in lwIP.

3.6. Developer notes

Applications should invoke `recv()` API repeatedly until it returns `EWOULDBLOCK` indicating there is no pending data in the receiving buffer. Otherwise, more and more incoming data will exhaust the buffer.

lwIP supports both blocking and non-blocking methods in the SDK. By default, sockets are blocking, indicating that if a socket call is issued, it cannot be completed immediately. For example, in the TCP three way handshaking the caller process is put to sleep waiting for the condition to be true. With non-blocking socket operation, the socket function call can return immediately and the application can proceed with other operations. Non-blocking I/O is suitable for single thread socket applications.

4. SSL/TLS: mbed TLS

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL) are cryptographic protocols designed to provide secure communication over the computer network. TLS and SSL use X.509 certificates and asymmetric cryptography to authenticate secure data communication and to negotiate the process with a symmetric session key that ensure message confidentiality.

mbed TLS offers libraries including SSL/TLS cryptographic communication capabilities for (embedded) devices and applications that provide end-to-end communication protection to the upper layer application protocols such as web browsing, email, instant messaging and voice-over-IP (VoIP).

Starting from the version 2.1.0, mbed TLS is released under Apache 2.0 License and enables developers to use mbed TLS in both open source and closed source projects.

4.1. Features

mbed TLS is an open source and commercial SSL library licensed under ARM Limited. This library easily integrates with new and existing (embedded) devices and applications and provides the building blocks for secure communication, cryptography and key management. Both the client-side and the server-side APIs support current SSL and TLS standards: SSL version 3.0, TLS version 1.0, TLS version 1.1 and TLS version 1.2. The cryptographic algorithms enabled in the SDK include:

1) Symmetric encryption algorithms: AES, Triple-DES (3DES), DES, ARC4.
2) Modes of operations: Cipher Block Chaining Mode (CBC).
3) Hash algorithms: MD5, SHA-1, and SHA-256.
4) RSA/PKCS#1 v1.5.
5) Random number generation: CTR_DRBG.

4.2. Memory usage

Airoha offers two configurations in the release, basic and mini. The basic package contains mandatory cipher suites supported by TLS v1.0 through TLS v1.2. The mini package includes minimal algorithm support for key exchange, cipher and hash algorithms. If the manufactures can control the deployment on both sides of the communication line, the memory footprint can be extremely reduced by manually selecting lightweight algorithms such as the algorithms using a pre-shared key.

Please refer to the sample configuration files `config-mtk-basic.h` and `config-mtk-mini.h` located under folder `<sdk_root>/middleware/third_party/mbedtls/configs/`.

Airoha IoT SDK provides a configuration file named `config-mtk-mini.h`. This is the minimum configuration to fulfill the requirement of TLS from versions 1.0 to 1.2. The `config-mtk-bsic.h` enables more commonly used cryptographic algorithms. The required ROM size for these two default configurations is shown in Table 6.

Table 6. Footprint of mbed TLS

Memory (Kbytes)	Basic		Mini	
	Debug	Release	Debug	Release
ROM	96	78	78	65
RAM	8.7	8.7	8.7	8.7
HEAP	26	26	26	26

Memory (Kbytes)	Basic		Mini	
(peak for one connection)				

4.3. Examples

mbed TLS is equipped with test cases. Developers can use them as a reference to develop and learn how to use the APIs. A few example applications can be found under `<sdk_root>/project/mt7687_hdk/apps/mbedtls/` directory.

4.4. Customization

The default configuration file for mbed TLS is `config-mtk-basic.h` in the SDK release. The file is located under `<sdk_root>/middleware/third_party/mbedtls/configs/config-mtk-basic.h`. Developers can provide their custom configuration file under `<sdk_root>/middleware/third_party/mbedtls/configs` and set the file name to the variable `MTK_MBEDTLS_CONFIG_FILE` in project's `feature.mk` makefile such as `<sdk_root>/project/mt7687_hdk/apps/iot_sdk/GCC/feature.mk`.

Define `MBEDTLS_DEBUG_C` in the configuration file to enable debugging and error handling.

You can switch the feature options in the configuration file and define some of the values or parameters. The configuration file is well documented and you can refer to the comments in `<sdk_root>/middleware/third_party/mbedtls/include/mbedtls/config.h`.

4.5. Limitations

Due to the limitation of ARM Cortex-M4 with floating point computational power, the TLS handshake at server side takes more than 10 seconds with RSA 2048-bit public key length. Therefore, the SDK disables the server option in the suggested configuration file.

4.6. Developer notes

1) In order to keep mbed TLS thread safe, it is important to keep a few things in mind.
2) Most functions use an explicit context. As long as the context is not shared among the threads, the application is thread safe. However, sometimes a context is shared indirectly. For example, an SSL context can point to an RSA context (the private key).
3) The rule of thumb is that a context should only be used or accessed by a single thread at a time, unless:
   a) The function is documented explicitly that it's thread safe to access the shared context, or
   b) You've applied an explicit locking mechanism, such as a mutex to protect a critical section through a wrapper function.
4) `MBEDTLS_SSL_MAX_CONTENT_LEN` is assigned with (6*1024) bytes in the example configuration files under `<sdk_root>/middleware/third_party/mbedtls/configs/config-mtk-*.h`, to reduce the heap usage. The default value of `MBEDTLS_SSL_MAX_CONTENT_LEN` is 16384 bytes. This value is defined in the specification. Modifying this value to reduce the required buffer may lead to interoperability problems. You can safely reduce this to a smaller size such as 2 Kbytes, if
   a) Both client and server sides support the `max_fragment_length` SSL extension (allowing reduction to less than 1k bytes for the buffer allocation).
   b) You control both sides of the connection or know the maximum size that will be sent in a single SSL/TLS frame.

5) Client verifies the server's identity with the received certificate during the handshake operation. Typically the received server certificate is composed of three-level hierarchy. If the length is larger, the TLS client will require more heap space for verification process.

6) Follow the steps below to configure the trusted root certificates.

   a) Since the SDK doesn't support File system in Airoha IoT Development Platform for RTOS, you can only keep the trusted certificate authentications (CAs) in memory and parse them by calling the `mbedtls_x509_crt_parse()` function.

   b) If there are several trusted CAs, invoke the `mbedtls_x509_crt_parse()` function several times to set them to the trusted CA chain one by one.

   c) Set authorization mode to `NONE`, `OPTIONAL` or `REQUIRED`, by calling the `mbedtls_ssl_conf_authmode()` function. If `REQUIRED` option is selected, the handshake operation will fail once the certificate verification fails.

   d) If the function `mbedtls_ssl_set_hostname()` is called, the client compares the hostname with the common name of the server certificate in the certification verification process.

5. HTTP (1.1) client: mbed HTTP Client

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Hypertext is a structured text that uses logical links (hyperlinks) between nodes containing text. HTTP is the protocol to exchange or transfer hypertext.

HTTP/1.1 is the most commonly used version of HTTP and was defined by RFC 2616 in 1999.

5.1. Features

HTTPClient implements the client-side of HTTP/1.1. It provides base interfaces to send HTTP requests and receive HTTP responses from a resource identified by a URI. It also supports HTTPS (HTTP over SSL/TLS) to provide secure communication.

5.2. Memory usage

The static footprint statistics for HTTP client are shown in Table 7, please note that the RAM size is 0 but it allocates the required buffer from system heap during the application execution.

Table 7. Footprint of the HTTP 1.1 client

	ROM (kB)	RAM, static analysis (kB)
Debugging disabled	3.5	0
Debugging Enabled	7.1	0

5.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/http_client/http_client_get/src/main.c`.
- Application Name: `http_client_get`.
- Application Overview: The application is a reference design on how to use `httpclient` API. The purpose is to connect to a website and retrieve results in a log. Simply call the function `httpclient_get()` to connect to the website server, send out a request and receive a response from the server. You can change the URL to connect to other websites and also the port to HTTPS to use the secure channel.
- File: `<sdk_root>/project/mt7687_hdk/apps/http_client/http_client_keepalive/src/main.c`.
- Application Name: `http_client_keepalive`.
- Application Overview: The application maintains a continuous connection with the HTTP server and sends requests periodically to it at a predefined interval. This example also shows how to set a customized HTTP header.
- File: `<sdk_root>/project/mt7687_hdk/apps/http_client/http_client_retrieve/src/main.c`.
- Application Name: `http_client_retrieve`.
- Application Overview: The application demonstrates how to download a large file by invoking the function `httpclient_retrieve_content()` repeatedly until it returns `HTTPCLIENT_RETRIEVE_MORE_DATA`.

5.4. Customization

Here are the configurable parameters for the HTTP Client.

- Set the macro HTTPCLIENT_DEBUG to 1 in `<sdk_root>/middleware/third_party/httpclient/inc/httpclient.h` to enable debugging.
- Switch MTK_HTTPCLIENT_SSL_ENABLE in the project configuration file (for example, in the GCC/feature.mk, or through the options in IDE menu of Keil or IAR) to enable SSL to secure HTTP messages.

5.5. Limitations

Currently only high level APIs are provided. Developers need to modify or use the internal functions in `<sdk_root>/middleware/third_party/httpclient/src/httpclient.c` to perform advanced configuration control over the HTTP link.

5.6. Developer notes

- The HTTP Client supports concurrent requests. However, the number of concurrent connections is limited by the heap space, socket configuration and DNS slots in lwIP.
- The peak heap usage for each HTTP over SSL connection process takes about 25 Kbytes for the certificate verification process during handshake.

6. XML: Mini-XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format, which is both human and machine-readable. It is defined by the W3C's XML 1.0 specification and by several other related specifications, all of which are free open standards.

Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services.

6.1. Features

The Mini-XML module is a small XML library that enables parsing the XML and XML-like data in the application without requiring large non-standard libraries. It supports reading of UTF-8 and UTF-16 and writing of UTF-8 encoded XML strings. Data is stored in a linked-list tree structure, preserving the XML data hierarchy and arbitrary element names, attributes and attribute values are supported with no preset limits, just available memory.

6.2. Memory usage

The XML library of the SDK has two build types, full and basic with optimized and reduced memory footprint.

The XML module provides several compilation options to disable unused sub-modules. Overall, the ROM size of most basic version is 13052 bytes. It is 21940 bytes if all of the sub-modules are enabled, see section 6.4, “Customization”.

6.3. Examples

1) File: `<sdk_root>/project/mt7687_hdk/apps/xml/src/main.c.`
2) Application Name: XML sample application.
3) Application Overview: The sample application is a reference to parse a typical XML file by means of this XML library. The developers can refer to this simple application and reuse the functions in their applications.

6.4. Customization

The XML module provides compile options to enable or disable sub-features. Table 8 shows the details.

Table 8. Options and footprint of XML

Compilation option	Function	ROM size (bytes)
MXML_SUPPORT_ENTITY	Convert the reserved characters in XML to corresponding character entities.	4584
MXML_SUPPORT_GET_FUNCTIONS	Get the attribute and content of an element	892
MXML_SUPPORT_SET_FUNCTIONS	Set the attribute and content of an element.	1040
MXML_SUPPORT_INDEX	Create the index with all elements, the elements are sorted by element name or attribute value.	1560
MXML_SUPPORT_SEARCH	Find the element by name.	812

6.5. Limitations

For the platforms that don't support file system, such as LinkIt 7687 development board, the file system related API sets of `mxml*Fd()` and `mxml*File()` are not accessible, and the XML objects can only be loaded and saved by `mxml*String()` APIs.

6.6. Developer notes

None.

7. JSON: cJSON

JSON (JavaScript Object Notation) is a lightweight data interchange format. It is convenient for humans to read and write and for the machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition. JSON is a text format that is completely language independent but uses conventions similar to C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.

JSON is built in two structures:

1) A collection of name or value pairs. In various languages, this is implemented as an object, record, structure, dictionary, hash table, keyed list, or associative array.
2) An ordered list of values. In most languages, this is implemented as an array, vector, list, or sequence.

7.1. Features

cJSON is an ultra-lightweight, portable, single-file, simple-as-can-be ANSI-C compliant JSON parser under MIT license. It's a single C source file along with its header file.

7.2. Memory usage

cJSON is a simple JSON parser with small footprint. The operational buffers are allocated from the system heap. The details are shown in Table 9.

Table 9. Footprint of cJSON

	ROM	RAM (static analysis)	Heap size
cJSON	8340	4	It varies depending on the parsed content, ranging from 800 bytes to 2200 bytes.

7.3. Examples

1) File: `<sdk_root>/project/mt7687_hdk/apps/cjson/src/main.c.`
2) Application Name: cJSON Test.
3) Application Overview: The application is a string parser using cJSON's APIs. The parsed string is stored in a `struct`. Developers can refer to this simple application and reuse the functions in their own applications.

7.4. Customization

None.

7.5. Limitations

Standard C `stdio.h` is not supported in the SDK, because ARM Cortex-M4 with floating point is a lightweight embedded system. The floating-point output is not as accurate as in a standard library. However, it is capable of handling common use cases such as representing longitude and latitude in geographic coordinate system.

7.6. Developer notes

1) `-fsingle-precision-constant` might cause an unexpected result when handling floating point data. The `-fsingle-precision-constant` compile option is used to improve performance due to less memory traffic by loading floating point constants in single precision format. This option also uses single precision constants in operations on double precision variables. For more information, please refer to semantics of floating point math in GCC Wiki.

   - For example, if the option is set while declaring a variable of type double, such as

```
double number = 987654321.23;
```

it will be compiled into `987654336`, for single floating point optimization.

8. SNTP: lwIP-contrib SNTP

Simple Network Time Protocol (SNTP), a less complex implementation of NTP (Network Time Protocol). It is used in embedded devices and in applications where high accuracy timing is not required. For more details about SNTP, please refer to RFC 4330.

8.1. Features

This SNTP implementation is derived from the `lwip-contrib` application projects, which can query the current time from a given server and set the acquired time to system clock. The SDK enables DNS server address, and multiple servers are supported in the release.

8.2. Memory usage

SNTP is a small utility in the lwIP package. The memory footprint is shown in Table 10.

Table 10. Footprint of SNTP

	ROM (Kbytes)	RAM, static analysis (bytes)
Debugging enabled	2	49
Debugging disabled	1	49

8.3. Examples

File: `<sdk_root>/project/mt7687_hdk/apps/sntp_client/src/main.c.`

1) Application Name: SNTP client.
2) Application Overview: The application is a reference usage of the SNTP. It receives GMT time from an SNTP server and sets system time by calling `sntp_init()` function and establishing connection with the SNTP server. It takes three seconds (`SNTP_RECV_TIMEOUT`) to establish the connection and it will update system time for every one hour (`SNTP_UPDATE_DELAY`).

8.4. Customization

1) The source configuration file is located under `<sdk_root>/middleware/third_party/sntp/src/sntp.c`. The header configuration file is located under `<sdk_root>/middleware/third_party/sntp/inc/sntp.h`. The parameters that can be customized or configured are shown below.
2) Define `SNTP_DEBUG`; use `LWIP_DBG_ON` to enable debugging; use `LWIP_DBG_OFF` to turn off debugging.
3) `SNTP_RECV_TIMEOUT` is SNTP receive timeout and the default value is three seconds. The default value of `SNTP_UPDATE_DELAY` is one hour, which is the update interval.
4) `SNTP_SET_SYSTEM_TIME` can be defined by user to set the current system time.
5) `SNTP_MAX_SERVERS` defines the number of SNTP servers to connect to. The current default value is five to indicate the maximum number of available servers in this SDK release. This value can be more than one. The larger the number, the more RAM the application needs.

8.5. Limitations

This implementation supports only unicast mode and doesn't support broadcast and multicast modes.

8.6. Developer notes

None.

9. DHCP daemon: Airoha minimal DHCPD

The SDK is enabled with a simple DHCP daemon to operate in soft access point (AP) mode to assign IP addresses to connected Wi-Fi station nodes.

9.1. Features

DHCPD is a program that operates as a daemon on a server to provide Dynamic Host Configuration Protocol (DHCP). The DHCPD offers IP address to the client.

9.2. Memory usage

DHCPD only responds with mandatory DHCP messages for IP address allocation for Wi-Fi-enabled devices, such as LinkIt 7687 HDK by SAC, turned into an Access Point. The footprint is extremely reduced versus a full functional DHCP server. See Table 11 for the required ROM/RAM size.

Table 11. Footprint of DHCP daemon

	ROM (bytes)	RAM, static analysis (bytes)
Debugging enabled	7431	77
Debugging disabled	3225	61

9.3. Examples

DHCPD is a standalone companion service designed to operate in soft AP mode. There is no example application developed with it.

9.4. Customization

1) Custom configuration can be implemented in `<sdk_root>/middleware/MTK/dhcpd/inc/dhcpd.h`.
2) Set `MTK_DEBUG_LEVEL` to `info` in `feature.mk` to enable debugging, or set `MTK_DEBUG_LEVEL` to none to disable debugging.
3) If DHCPD_SAVE_CLIENT_CONFIG_ON_LINE is defined, the allocation information such as IP and MAC addresses of the clients will be stored in the RAM. The pre-allocated IP will be allocated preferably to the client, if the IP is available. If DHCPD_SAVE_CLIENT_CONFIG_ON_LINE is not defined, the first available IP will be allocated to the client.

- For example, the pre-allocated IP for client A may be allocated to client B when only the IP is available. In this case, allocation information for client A will be lost. After reboot, allocation information will be lost, too.

4) The DHCPD configuration, such as server IP, gateway, DNS and more, is done through the parameter settings of `dhcpd_start()`. For more details on the function prototype, refer to the header file `dhcpd.h`.

9.5. Limitations

This is a simple daemon and provides limited support to the message types from DHCP clients. It always replies `DHCPOFFER` to any `DHCPDISCOVER` message and `DHCPACK` to `DHCPREQUEST` messages, and sends a `DHCPNAK` message for `DHCPREQUEST` to request for an invalid IP address.

9.6. Developer notes

DHCPD does not actually maintain a lease timer to manage the database of assigned IP addresses. Instead, it uses a more efficient approach by using the knowledge from soft AP module about the absence of client nodes. The DHCP server removes the assigned IP address immediately responding to the client node disconnection notification message.

10. MQTT client: Paho Embedded MQTT C/C++ Client

MQTT is a lightweight publish/subscribe messaging protocol, originally created by IBM and Arcom (later to become part of Eurotech) around 1998. More information about the protocol can be found on the mqtt.org community website.

10.1. Features

The Paho project is created to provide scalable open-source implementations of open and standard messaging protocols aimed at new, existing and emerging applications for M2M and IoT devices and applications. Paho reflects the inherent physical and cost constraints of device connectivity. Its objectives include effective levels of decoupling between devices and applications, designed to keep markets open and encourage the rapid growth of scalable Web and Enterprise middleware and applications. Paho supports MQTT publish/subscribe client implementations on embedded platforms, along with corresponding server support as determined by the community.

In this release the Paho MQTT library is integrated with mbed TLS to widen the connectivity support of various types including:

- Unencrypted.
- Encrypted, CA certificate required.
- Encrypted, CA/Client certificate required

A new porting layer is designed to port the library to Airoha IoT Development Platform for RTOS, based on mbed TLS, lwIP and Airoha Hardware Abstraction Layer (HAL) API. The porting architecture is shown in Figure 3. For each MQTT application a network connection is established using `ConnectNetwork()` or `TLSConnectNetwork()` APIs that are implemented in the platform-dependent porting layer. Then use `MQTTClient()` to initiate a MQTT client instance, followed by `MQTTConnect()`, `MQTTSubscribe()` or `MQTTPublish()` to interact with servers. The MQTT*() APIs implemented in the green blocks in the diagram are platform independent.

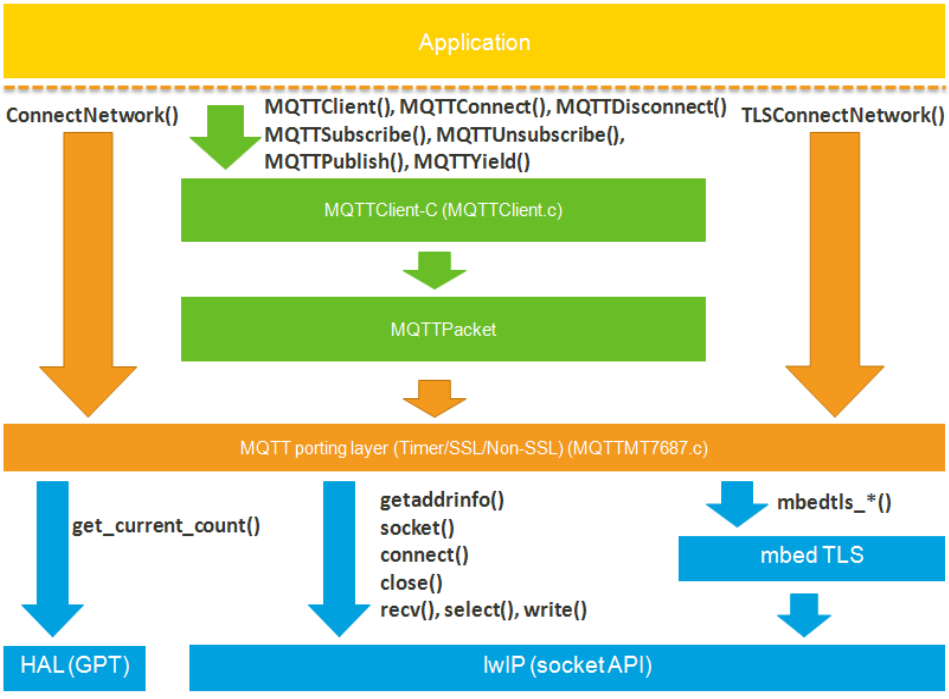

Figure 3. Architecture of the MQTT integration

10.2. Memory usage

MQTT was originally designed to be lightweight transport protocol for M2M communication, see Table 12 for the required ROM/RAM size.

Table 12. Footprint of the MQTT client

	ROM (bytes)	RAM, static analysis (bytes)
Enable Debugging	12659	64
Disable Debugging	8521	64

10.3. Examples

File: `<sdk_root>/project/mt7687_hdk/apps/mqtt_client/.`

Application Name: MQTT Client.

Application Overview: The application is based on the MQTT functions implementing connectivity with an unencrypted or encrypted MQTT server. Developers can refer to this simple application and re-use the functions in their own applications.

10.4. Customization

Use `MT76XX_MQTT_DEBUG` to enable or disable the logs. Define it in `<sdk_root>/middleware/third_party/mqtt/MQTTClient-C/src/mediatek/MQTTMediatek.h`.

10.5. Limitations

None.

10.6. Developer notes

None.

11. HTTP server: axTLS HTTPD

HTTP daemon (HTTPD) is an information technology that processes requests through HTTP. The term can refer either to the entire computer system, an appliance or to specific software that accepts and supervises the HTTP requests.

11.1. Features

axTLS HTTPD is a small HTTP server. It provides base interfaces to start a server and stop a server. The server supports GET and POST HTTP methods. It also provides basic authentication and CGI functionality.

11.2. Memory usage

The static footprint statistics for HTTP server is shown in Table 13, please note that the RAM size is 0 but it allocates the required buffer from system heap during the application execution.

Table 13. Footprint of the HTTP server

	ROM (bytes)	RAM, static analysis (bytes)
Enable debugging	21561	593
Disable Debugging	13310	593

11.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/httpd/src/main.c.`
- Application Name: `httpd.`
- Application Overview: `httpd` is a reference application based on the HTTPD API usage. It implements functions to initialize, start and stop a HTTP server.

11.4. Customization

1) The username and password (e.g. admin/admin) can be configured in the authentication settings with the `user_config()` API located under `<sdk_root>/middleware/third_party/httpd/src/auth_check.c.`
2) The maximum number of supported connections is five. Developers can use `INITIAL_CONNECTION_SLOTS` option to redefine it in `<sdk_root>/middleware/third_party/httpd/inc/axhttp.h.`
3) In addition, the server provides custom configuration options in `<sdk_root>/middleware/third_party/httpd/inc/config.h.`

- `HTTPD_DEBUF`: define HTTPD_DEBUF to enable debugging.
- `CONFIG_HTTP_DIRECTORIES`: list all files when client accesses the default HTML and server does not have a default HTML.
- `CONFIG_HTTP_HAS_IPV6`: define `CONFIG_HTTP_HAS_IPV6` to support IPv6 address.
- `CONFIG_HTTP_PORT`: define a default server port.
- `CONFIG_HTTP_HOME_HTML`: define a default HTML.

11.5. Limitations

The HTTP server currently does not support encrypted connection over TLS.

11.6. Developer notes

The default value of `INITIAL_CONNECTION_SLOTS` is 6, which allows five concurrent connections with one additional socket to list the incoming sockets. The maximum number of concurrent connections is limited by this macro and the total number of TCP sockets is defined in the underlying TCP module.

12. HTTP/2 client: nghttp2

HTTP/2 is an alternative to HTTP1.1. HTTP methods, status codes and semantics are the same and it should be possible to use the same APIs as HTTP/1.x (possibly with some small additions) to represent the protocol.

The focus of the protocol is on performance; specifically, end-user perceived latency, network and server resource usage. One major goal is to allow the use of a single connection from browsers to a website.

The basis of the work was SPDY, but HTTP/2 has evolved to take the community's input into account, incorporating several improvements in the process.

12.1. Features

The framing layer of HTTP/2 is implemented as a reusable C library. On top of that, the SDK includes implementations of an HTTP/2 client, server, proxy, load test and benchmarking tools for HTTP/2 and SPDY.

An HPACK encoder and decoder are available as public APIs.

An experimental high level C++ library is also available.

It only enables below features:

- The C library of the HTTP/2 framing layer.
- HPACK encoder and decoder.

12.2. Memory usage

HTTP/2 is a new protocol and therefore it requires more resources in terms of code and heap size. See Table 14 for the required ROM/RAM and heap size.

Table 14. Footprint of the HTTP/2 client

	ROM (bytes)	RAM, static analysis (bytes)	Heap size (1 connection)
nghttp2	60545	1748	80kB

12.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/nghttp2_client/src/main.c`.
- Application Name: `nghttp2client_test`.
- Application Overview: The application is a reference to usage of mbed TLS API and nghttp2 to connect with HTTP/2 server. If the server does not support HTTP/2, it will fall back to HTTP/1.1. Developers can refer to this simple application and re-use the functions in their own applications.

12.4. Customization

1) Please refer to the file
2) `<sdk_root>/middleware/third_party/mbedtls/configs/config-mtk-http2.h` to enable 1.2 version of TLS and Application Layer Protocol Negotiation (ALPN) for mbed TLS.

12.5. Limitations

nghttp2 is unable to negotiate next protocol with SPDY servers because the underlying mbed TLS does not support Next Protocol Negotiation (NPN). NPN was later replaced with a reworked version, ALPN supported by mbed TLS.

12.6. Developer notes

None.

13. File System: FatFs

FatFs is a generic FAT file system that manages the access to storage devices for small embedded systems. FatFs is written in compliance with ANSI C (C89) and completely separated from the disk I/O layer.

13.1. Features

To facilitate the data (files or directories) management in the storage device, FatFs provides the following features:

- Windows OS compatible FAT file system.
- Platform independent, easy to port to any target platform.
- Small footprint for code and work area (file system objects, file objects, etc).
- Long file name support in ANSI/OEM or Unicode.
- RTOS support for multi-tasking.
- Multiple sector size support up to 4kB.
- Read-only, optional API, I/O buffer and other features.

More information on the FatFs and its features can be found here.

13.2. Memory usage

The memory usage varies depending on the configuration options, as described in section 13.4, “Customization”. The FatFs module provides several compilation options to disable unused sub-modules. Overall, according to the configuration in the SDK release, the ROM size of the current release is 12466 bytes, the RAM size is 518 bytes. The memory size may be different for different versions of the SDK.

13.3. Examples

The source code and API documentation of the FatFs can be found at official website.

13.4. Customization

Users can customize the configuration for the FatFs in `ffconf.h` header file. The configurable parameters include the volume to support, the sector size, etc.

More details can be found in the online documentation.

13.5. Limitations

The list of feature limitations for the FatFs is provided below.

- Multiple sector size support up to 4kB.
- FAT sub-types - FAT12, FAT16 and FAT32.
- Number of open files - unlimited depending on the available memory.
- Number of volumes - up to 10.
- File size - up to 4GB minus 1 byte. (FAT specifications.)

- Volume size - up to 2TB at 512 bytes per sector. (FAT specifications.)
- Cluster size - up to 64kB at 512 bytes/sector. (FAT specs.)
- Sector size - 512, 1024, 2048 and 4096 bytes. (FAT specs.)

13.6. Developer notes

FatFs integrated in current SDK version only supports file system on the SD (Secure Digital Memory Card) or eMMC (Embedded Multi Media Card) without flash.

- FatFs is not enabled by default in order to provide user with the flexibility of choosing whether to use this file system or other file systems.

14. CMSIS

The CMSIS is a vendor-independent hardware abstraction layer for the Cortex-M processor series and defines generic tool interfaces. The CMSIS enables consistent device support and simple software interfaces to the processor and the peripherals, real-time operating systems and middleware components simplifying software re-use, reducing the learning curve for microcontroller developers and reducing the time to market for new devices.

The CMSIS is intended to enable the combination of software components from multiple middleware vendors.

14.1. Features

The CMSIS components are:

- CMSIS-CORE. Implements basic run-time APIs for a Cortex-M device and provides convenient access to the processor core and the device peripherals.
- CMSIS-DSP. This library provides a suite of common signal processing functions to apply on Cortex-M processor based devices.

Airoha IoT development platform only includes CMSIS-CORE and CMSIS-DSP.

14.2. Memory usage

CMSIS-CORE APIs are implemented in header files, so the memory usage is added to the source file. The library size of the CMSIS-DSP is up to 5.4MB and the actual memory usage depends on which and how many APIs are in use.

14.3. Examples

CMSIS API examples can be found at official website.

14.4. Customization

No customization or configuration is required to use CMSIS-CORE and CMSIS-DSP, apply them directly in your implementation..

14.5. Limitations

None.

14.6. Developer notes

Airoha IoT Development Platform only includes CMSIS-CORE and CMSIS-DSP.

- It’s recommended to access ARM official website to find more detailed information about CMSIS.

15. LZMA: LZMA Decoder

The LZMA is an algorithm performing lossless data compression. The LZMA SDK provides a high compression ratio and fast decompression. The Airoha IoT SDK only uses the LZMA decoder algorithm.

15.1. Features

The LZMA decoder provides functions to decompress the compressed data encoded by LZMA encoder.
LZMA decoder has the following features:

- Small memory requirements: 8 to 32kB + Dictionary Size.
- Small code size: 2 to 8kB, depending on speed optimizations.

More information on the LZMA and its features can be found here.

15.2. Memory usage

The memory usage is shown in Table 15.

Table 15. LZMA decoder memory usage

	ROM (bytes)	RAM, static analysis (bytes)
Enable Debugging	7182	40*1024 (suggested decoding buffer)
Disable Debugging	6708	40*1024 (suggested decoding buffer)

15.3. Examples

LZMA decoder module is used to decode FOTA package file in bootloader only. Because Cortex-M4 binary size is usually too large to have enough buffer for the whole file data, there is a wrapper function `lzma_decode2flash()` provided in `<sdk_dir>/middleware/third_parity/lzma_decoder/inc/lzma_decoder_interface.h` header file to decode data block by block, then call HAL flash API to write the output data to the specified address on NOR flash.

More information on the LZMA decoder interface can be found in the bootloader module's source file (`<sdk_dir>/driver/board/mt25x3_hdk/bootloader/core/src/bl_fota.c`).

15.4. Customization

None.

15.5. Limitations

None.

15.6. Developer notes

LZMA decoder integrated in current version of the SDK only supports bootloader decoding.

16. mDNS server: mDNSResponder

Multicast DNS (mDNS) provides the ability to perform DNS-like operations on the local link in the absence of any conventional Unicast DNS server. DNS-based Service Discovery (DNS-SD) specifies how DNS resource records are named and structured to facilitate service discovery.

The mDNSResponder project enables to perform DNS-like operations based on mDNS protocol and DNS-SD.

The focus of mDNSResponder in the SDK is to publish services using mDNS.

16.1. Features

An mDNS daemon and server are available as public APIs.

Please refer to the header files `mdns.h` and `dns_sd.h` located under folder `<sdk_root>/middleware/third_party/mDNSResponder/inc/`.

The following features are enabled:

- Start and stop the mDNS daemon.
- Publish mDNS services.
- Update and unregister an existing service.
- Responding.

16.2. Memory usage

See Table 16 for the required ROM/RAM and mDNS daemon task stack size details.

Table 16. Footprint of the mDNSResponder

	ROM (bytes)	TCM-RAM, static analysis (bytes)	mDNS stack size
mDNSResponder	86361	11468	15kB

16.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/mdns_publish_service/src/main.c`.
- Application Name: `mdns_publish_service`.
- Application Overview: This is a reference application to publish a service using mDNSResponder APIs. Developers can reuse the functions in their own applications.

16.4. Customization

None.

16.5. Limitations

The mDNSResponder does not support mDNS client features, such as discovery, resolution and cache.

16.6. Developer notes

None.

17. WebSocket: librws

WebSocket protocol is a TCP-based protocol, which provides full-duplex communication channels for the server-client communication. Unlike HTTP, the WebSocket server can send content without being solicited by the client. The new URI schemes, ws and wss, are defined for WebSocket unencrypted and encrypted connections, separately.

Librws is a small cross-platform WebSocket client library in C, released under the MIT license. Airoha provides an improved secure WebSocket based on mbed TLS.

17.1. Features

Librws provides the following features:
No additional dependencies

Single header library interface `librws.h` with public methods

Thread safe

Send/receive logic in background thread

More information on the librws and its features can be found here.

17.2. Memory usage

The memory usage is shown in Table 17.

Table 17. Librws memory usage

	ROM (bytes)	RAM, static analysis (bytes)
Disable secure WebSocket	8843	20
Enable secure WebSocket	10822	20

17.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/websocket_client/src/main.c`.
- Application Name: `websocket_client`.
- Application Overview: The application is a demonstration of the WebSocket client. The developers can reuse the functions in their own applications.

17.4. Customization

- Airohaprovides an implementation to support the secure WebSocket based on mbed TLS. To enable secure WebSocket, mbed TLS module should be included and "MTK_WEBSOCKET_SSL_ENABLE = y" should be added into the GCC project makefile, such as `feature.mk`.
- The file `config-mtk-websocket.h`, is used for the secure WebSocket as the mbed TLS configuration file, by default. Developers can customize and use their own mbed TLS configuration files based on the specific requirements of the WebSocket servers.

17.5. Limitations

None.

17.6. Developer notes

None.

18. CoAP: libcoap

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks in the IoT and Machine-to-Machine devices. The CoAP is designed to adapt to resource-constrained, energy efficient and lossy internet devices. It's a UDP-based protocol providing a request/response interaction module between constrained devices. It can be easily translated into HTTP, however, it keeps the message overhead small.

18.1. Features

`libcoap` provides the following features:

- Asynchronous Messaging
- Block Transfer
- Clock Handling
- Option Filters
- Resource observation
- URI Parsing Functions

More information on the libcoap and its features can be found here.

18.2. Memory usage

The memory usage is shown in Table 1Table 18.

Table 18. `libcoap` memory usage

	ROM (bytes)	RAM, static analysis (bytes)
`libcoap`	12896	0

18.3. Examples

- File: <sdk_root>/project/mt7687_hdk/apps/coap/src/main.c.
- Application Name: coap.
- Application Overview: The application is a demonstration of the CoAP client and server interaction. The developers can reuse the functions in their own applications.

18.4. Customization

All feature options, including `COAP_RESOURCES_NOHASH` and `WITH_POSIX`, can be defined in `<sdk_root>/middleware/third_party/libcoap/include/coap/config.h`.

18.5. Limitations

None.

18.6. Developer notes

None.

19. AWS IoT

AWS IoT is a managed cloud platform to securely connect devices with cloud applications and other devices. AWS IoT can support billions of devices and trillions of messages, and can process and route those messages to AWS endpoints and to other devices reliably and securely. With AWS IoT, applications can keep track of and communicate with all your devices, all the time, even when they aren't connected.

19.1. Features

AWS IoT makes it easy to use AWS services like AWS Lambda, Amazon Kinesis, Amazon S3, Amazon Machine Learning, Amazon DynamoDB, Amazon CloudWatch, AWS CloudTrail, and Amazon Elasticsearch Service with built-in Kibana integration, to build IoT applications that gather, process, analyze and act on data generated by connected devices, without having to manage any infrastructure.

More information on the AWS IoT and its features can be found here.

19.2. Memory usage

The memory usage is shown in Table 19.

Table 19. AWS IoT memory usage

	ROM (bytes)	RAM, static analysis (bytes)
AWS IoT	27904	1

19.3. Examples

- File: `<sdk_root>/project/mt7687_hdk/apps/aws_iot/src/main.c`.
- Application Name: `aws_iot`.
- Application Overview: The application is a demonstration of the usage of AWS IoT SDK.

19.4. Customization

- Set `MTK_MBEDTLS_CONFIG_FILE = config-aws-iot.h and AWS_IOT_SUPPORT = y` in your project.
- Get certificate from AWS IoT first from here, then set the certificate in `<sdk_root>/middleware/third_party/aws_iot/aws_iot_cert_rtos.h`.

19.5. Limitations

More information on the AWS IoT's limitations can be found here.

19.6. Developer notes

None.