WLAN Controller design specifications document

Wireless Lab

IIT Bombay

Project under Prof. Abhay Karandikar

Documented by Navneet Agrawal

[Navneet.agrawal@iitb.ac.in](mailto:Navneet.agrawal@iitb.ac.in)

|  |  |  |
| --- | --- | --- |
| **Dated** | **Updates** | **Modified by** |
| 01/06/2015 | Document created | Navneet Agrawal |
|  |  |  |

OpenCAPWAP  
Software Specifications

Contents

[Introduction 2](#_Toc420609304)

[Overview 2](#_Toc420609305)

[Aim of the project 2](#_Toc420609306)

[CAPWAP Protocol Implementation (OpenCAPWAP) 2](#_Toc420609307)

[OpenCAPWAP – Design specifications 2](#_Toc420609308)

[State Machine 2](#_Toc420609309)

[State Machine implementation in OpenCAPWAP 2](#_Toc420609310)

[Working 4](#_Toc420609311)

[CAPWAP Protocol implementation 6](#_Toc420609312)

[AC- WTP communication at different states 7](#_Toc420609313)

[External Application paradigm 9](#_Toc420609314)

[AC side Applications 9](#_Toc420609315)

[WTP side Applications 9](#_Toc420609316)

[Compilation and running AC/WTP daemon 10](#_Toc420609317)

# Introduction

Communication protocols, CAPWAP, are designed to make the Access Controller interoperable with any CAPWAP compatible Access Point. The communication channel is secured using DTLS (Data Tunneling Layered Security) protocols.

For detailed specifications and working of CAPWAP, please refer to RFC-5415 and OpenCAPWAP paper.

OpenCAPWAP Paper: <http://www.sciencedirect.com/science/article/pii/S1389128608003095>

RFC 5415: https://tools.ietf.org/html/rfc5415  
RFC 5416: https://tools.ietf.org/html/rfc5416

# Overview

## Aim of the project

Recent increase in demand of mobile communication data, telecom service vendors are working towards offloading mobile data traffic using WLAN network which works on the free unlicensed ISM band. The idea is to provide city-wide WiFi network to all users, enabling a seamless offloading of data between mobile networks (3G/4G/LTE) and WiFi networks. This will not only reduce the burden on core mobile networks, but also provide better services to the users.

## CAPWAP Protocol Implementation (OpenCAPWAP)

OpenCAPWAP is developed by Vellore (Bernaschi et all). OpenCAPWAP is a user-space program which implements CAPWAP communication protocols between an Access Controller (AC) and Wireless Termination Points (WTP). The program is written in Object C.

The scope of this document does not include specifications of the CAPWAP Protocols. Hence, we will only discuss important design aspects of OpenCAPWAP and modifications implemented by our team.

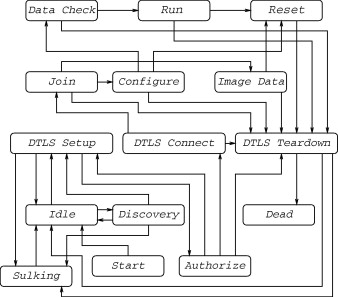
# OpenCAPWAP – Design specifications

## State Machine

CAPWAP state machine enables active and secured communication between AC and WTP. State information of both AC and WTP is being kept by the state machine. Different levels of state require different communication exchanges.

### State Machine implementation in OpenCAPWAP

Major part of the CAPWAP protocol is the implementation of the State Machine. All communication is dependent on the state of AC and WTP. Hence it is important that we know how this state machine is implemented in the software before discussing the design variations.



The state diagram reported in figure above represents the lifecycle of a WTP-AC session with a Finite State Machine (FSM), as defined in the protocol specification [RFC 5415]. Use of DTLS by the CAPWAP protocol results in the juxtaposition of two nominally separate yet tightly bound state machines. The DTLS and CAPWAP state machines are coupled through an API consisting of commands and notifications. Certain transitions in the DTLS state machine are triggered by commands from the CAPWAP state machine, while certain transitions in the CAPWAP state machine are triggered by notifications from the DTLS state machine.

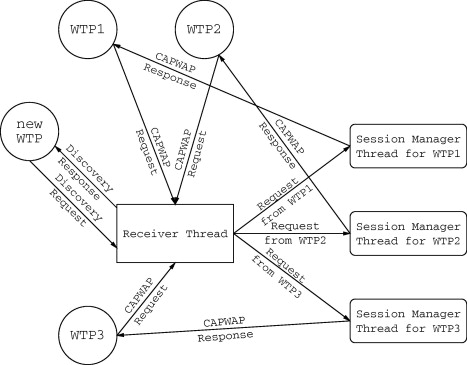
The same FSM is defined for both the WTP and the AC, although some states and transitions are implemented only on either the WTP or the AC. A complete description of the FSM is reported in the Section 2.3 of [RFC 5415]. A normal session for the WTP begins with the Start state. After the initialization is complete, the WTP enters the Idle state. The WTP then proceeds to the Discovery phase to find an AC to connect with. The Discovery state is skipped if the WTP is instructed to try to connect to a fixed AC. In both cases the WTP moves to the DTLS Setup phase, to establish a secure DTLS connection with the AC. A transition from DTLS Setup to Authorize occurs when the DTLS session is being established, and the DTLS stack needs authorization to proceed with the session establishment. If all goes well, the WTP enters the DTLS Connect state, and when the DTLS connection is established the Join state is reached. In the Join state the AC and the WTP start communicating with each other and the CAPWAP session can begin. Upon the reception of a successful Join Response message from the AC, the WTP is instructed to either download new executable firmware (in which case it enters the Image Data state, and the session ends because the device is reset) or it is configured through the appropriate messages sent by the AC (in the Configure state). If the success of the configuration process is confirmed by messages exchanged in the Data Check state, the WTP eventually reaches its normal operating state, represented in the FSM by the Run state. There are a number of possible events that may occur in the Run state, the most relevant is the Configuration Update Request by which the AC requires the WTP to modify its configuration.

The other states illustrated in the FSM have intuitive meanings. The Sulking state is somehow a special case that is defined for situations in which the WTP cannot communicate with an AC. The WTP exits from the Sulking state and starts a new Discovery phase (after a temporary transition to the Idle state) when the SilentInterval timer expires.

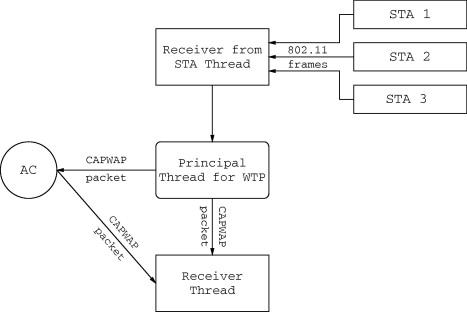
## Working

Our implementation of the CAPWAP protocol consists of two Linux applications, one running on the ACs and one running on the WTPs. Each WTP acts as a client of an AC. Each AC manages both the communication with WTPs already registered and new requests sent by WTPs not-yet registered.

For both applications we followed a multi-threaded programming model which represents a reasonable trade-off between modularity and efficiency. We describe the structure of our implementation starting from the AC component. In the beginning, there is a single receiver thread that is in charge of receiving any packet arriving from the WTPs. When a packet arrives, the receiver thread extracts the source address and checks if it corresponds to an already established session. In this case the packet is queued to a list of pending requests associated with that session. If it is a request coming from an unknown WTP, the receiver thread analyzes the message to determine if it is a Discovery Request. In this case, it sends in reply a Discovery Response message. The reason why it is the receiver thread that directly manages the Discovery Request messages is that it is not worth starting a new thread unless the WTP continues the registration procedure (a WTP may send Discovery Request messages in broadcast so that more than one AC receives them). If the message is a Client Hello, that is the first message sent by a WTP when it wants to establish a DTLS session with the AC, a new session manager thread is created. The session manager thread replies to the message and manages any future request (including the sending of possible responses) coming from the same WTP. The flow of messages among WTPs and AC threads is shown in Figure below.



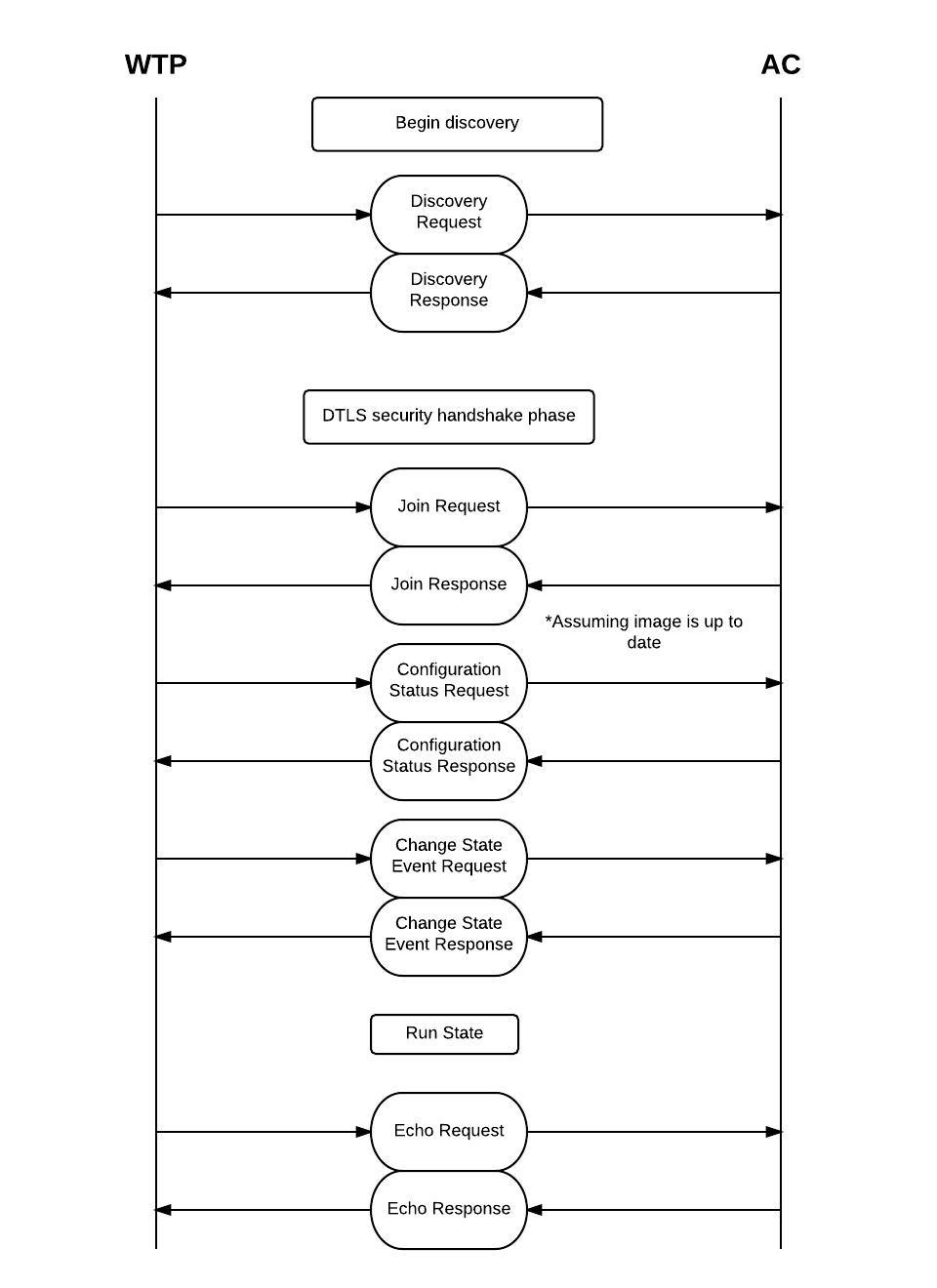
The number of threads used on the WTP is always limited to three. During the Discovery phase, there is only one principal thread in charge of the communication with potential ACs. After receiving the Discovery Response messages and having chosen one of the ACs, another receiver thread is created. This thread sends to the selected AC the Client Hello message to establish the DTLS session, but this is the only message sent by this thread. All other requests are sent by the principal thread which also manages the responses received by the AC. The need for an independent thread dedicated exclusively to the reception of packets arises because during the Run phase the AC may send its own requests to a WTP. The receiver thread shares packets with the principal thread using a list as in the AC case. The third, receiver-from-STA, thread is required for intercepting packets sent by the STAs connected to the WTP. This operation is necessary because some packets, in particular the Association Requests, sent by the STAs may have to be forwarded to the AC (see Section 3.1). The relation among the three threads of the WTP application is shown in Figure below.



## CAPWAP Protocol implementation

The CAPWAP protocol, like other existing control and management protocols (e.g., the SNMP), relies on UDP as transport protocol. Since UDP does not guarantee reliable communications, we implemented, as required by the CAPWAP protocol, a retransmission mechanism for requests whose response does not arrive within a specified timeout. As to the security requirements imposed by the CAPWAP protocol, our implementation resorts to the OpenSSL implementation of the DTLS protocol to fulfill them. In case of timeouts and consequent re-transmission of packets it is necessary to cipher them again to avoid that the receiver could assume that a replay attack is in progress.

All code is organized as a set of modules to make easier the replacement of single components. This is a fundamental requirement for the implementation of a protocol which is not completely and definitely specified yet. From the very beginning, we included a logging mechanism with multiple levels of verbosity to ease troubleshooting activities.

The state machine is implemented as follows:

### AC- WTP communication at different states

Let’s start with WTP discovery state.

Main file of WTP is WTP.c. In the main() function, various states of WTP are maintained and recorded.

1. Discovery State (WTPDiscoveryState.c)

Function: CWEnterDiscovery()

This function takes care of discovery state communication between AC and WTP. WTP checks its config file for the list of ACs and then assembles a discovery request to send to each AC.

CWAssembleDiscoveryRequest()

Finds IP address of AC

WTP Enters Discovery State

Sends the request to the AC

Assemble Discovery Request Message

Discovery request message is a CAPWAP control message. Hence it has to be wrapped in control headers. Inside control header wrapper, there are different message elements which are included in the Discovery request. Refer to RFC5415 for more details.

1. Discovery Type message element
2. WTP Board Data
3. WTP Frame Tunnel Mode
4. WTP MAC Type
5. Discovery Response

After WTP sends a discovery request, AC receives the request and sends a response back to the IP of WTP.

Assemble Discovery Response

Parse Discovery Request

AC receives Incoming Packet

Send response to WTP

AC manages incoming packet in file ACMainLoop.c. The function used to manage incoming packets is *CWACManageIncomingPacket()***.**

1. DTLS session initialization

WTP will enter the DTLS security state after successfully receiving the Discovery Response from AC. WTP will initiate the DTLS security session and send security keys to AC. A new thread is created to receive DTLS packets.

Create DTLS packet receive Thread

WTP starts DTLS Security session

WTP enters DTLS security state

WTP sends security keys to AC

AC creates a new thread to manage WTP

AC receives the Security information

AC authorizes the WTP and setup DTLS security tunnel

This process takes place in WTPJoinState.c file. The function handling this process is *CWWTPEnterJoin()*.

1. Join State

WTP, after successfully receiving the security information from AC, enters the Join State. WTP assembles the Join Request message and sends it to AC. AC responds with Join Response message. Inside the *CWWTPEnterJoin()* function, WTP calls for *CWWTPSendAcknowledgedPacket()* which assembles join request, receives response, parse the response parameters and saves the new values in WTP.

1. Configure State

WTP enters the configure state after successful Join state. Again WTP sends the configure request to AC and parse and saves the response. The function *CWWTPEnterConfigure()*  in WTPConfigureState.c handles the process here.

AC sends the appropriate configuration to WTP and WTP saves configuration parameters.

1. Change state event

If WTP change its state after the configure state, it sends a change state event request. AC responds to this request and acknowledges the change in state of WTP.

1. Run state

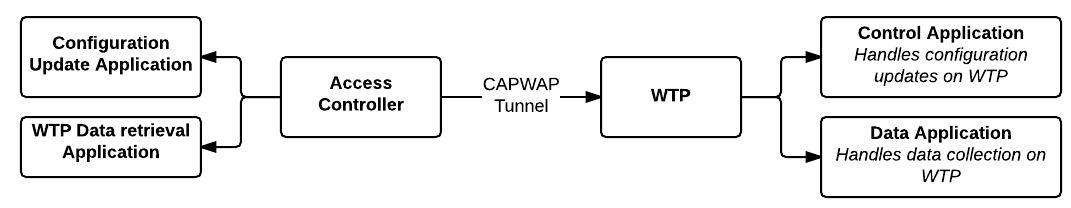
Finally, WTP enters the run state. In run state, WTP enters a loop where it continuously check for any incoming requests from the AC. The function *CWEnterRun()* in WTPRunState.c manages the process in this stage.

There are two different kinds of packets that are received on WTP.

1. Packet received from AC : these are the packets sent by AC to WTP. When WTP receive packets from AC, it calls the function *CWWTPManageGenericRunMessage()*. This function checks for different types of control messages and sends them to specific functions for parsing the information.
2. Frames send by Station server: The frames received by WTP from the station server are encapsulated in a CAPWAP data message and sent to AC. AC will extract the IEEE 802.11 frames from this packet and take necessary actions.

## External Application paradigm

At the current stage of development we assume that each WTP has a single radio component. Moreover both Data and Control CAPWAP messages use the same port although the protocol expects that, in the Data Check phase, WTP and AC negotiate a new communication channel. Actually, such choice is due mainly to the uncertainty that it is still present within the CAPWAP working group about the possible benefits of this feature. Finally, in the Run state all the CAPWAP messages are currently managed. Among the others, the Configuration Update Request is fully supported, allowing for the management of IEEE 802.11 WTP Quality of Service messages that have been used to test the implementation.



### AC side Applications

Two components of AC side applications are –

1. WTP Data retrieval Application:

This application will run on AC to retrieve the information data from different WTPs. AC will receive information data periodically from all connected WTP. The data packet will contain WTP-ID to identify which WTP has sent the data. This data will be updated on the Database connected with the Network Management System (Web UI). Present implementation only processes the connected Stations information from every WTP.

Details regarding this application are explained in the next section.

1. Configuration Update Application:

Configuration Update Application will send the Configuration Update request to a specific WTP. This application will be linked with the Network Management System (Web UI) so that the network administrator can configure any WTP connected to the AC. Present implementation only includes **Channel** and **TxPower** configuration.

Details regarding this application are explained in the next section.

### WTP side Applications

Two components of WTP side applications are –

1. Control Application:

Control Application will handle the configuration update requests from AC. When WTP main daemon receives a packet containing the configuration update request message, it will forward this message element to control application. The control application will read the information and implement it accordingly. Presently control application only handles Channel and TxPower implementations.

Details regarding this application are explained in the next section.

1. Data Application:

This application collects information related to radio-resource management. The collected information is formatted into a proper structure and sent to AC as a data message. The data application runs as a separate process on WTP and sends this information periodically.

Details regarding this application are explained in the next section.

## Compilation and running AC/WTP daemon

OpenCAPWAP compilation require 32 bit kernel. Also, for implementing DTLS security, it uses OpenSSL library. Once all the libraries are in place, simple *make* command will compile the code.

There are setting files for configuring WTP and AC.

**Setting files**

WTP config – config.wtp  
This file contains list of information for WTP configuration. Set the value of AC\_ADDRESS to AC IP Address in this file

WTP settings – setting.wtp.txt  
Set the wlan interface name in this file.

AC config – config.ac  
This file has configuration values for AC. In general default values are set in this and nothing has to be changed.

AC setting – setting.ac.txt   
This file has QoS value parameter information values.  
 **Installation**

To install AC and WTP, just run *make*  from inside the OpenCAPWAP folder

To install APPs, run *make* from inside the APP folders.

**Running WTP and AC**

To run to code, you have to have root privileges.   
For AC run– “*sudo ./AC .”*For WTP run – “*sudo ./WTP .”*

**Running APPs**

*WTP Side – Netlink  
$> ./NetlinkServer*

*WTP Side – WTPConfigUpdateServer  
$> ./WTPConfigUpdateServer*

*AC Side – ACDataInfoClient (for station info)  
$> ./ACDataInfoClient*

*AC Side – WUM (Ani)  
$> ./wum –w <WTP ID> -c <Channel> -t <TxPower>*

**Log files**

Log files are created at *“/var/log/<ac/wtp>.log.txt”*

## AC configuration update application

The configuration update application in AC is called WUM (WTP Update Manager). This application runs as a client on the AC. WUM client connects to AC side server (CWManageApplication) on port 1235.

AC daemon creates a separate thread called CWManageApplication (file: ACInterface.c). This thread creates a server on the port 1235 and listens for incoming client applications. There are three kind of features supported via client Application.

QUIT\_MSG: When client send this message, CWManageApplication server disconnects the client.

LIST\_MSG: The server sends the list of connected WTP to the client application.

CONF\_UPDATE\_MSG: WUM sends the conf update message to the server. The server identifies the configuration update message and sends it to specific WTP.

### Structure of CONF\_UPDATE\_MESSAGE:

* WTP Index : 4 Bytes – Integer
* Message Elements Count – 4 Bytes – Integer
* Message Elements Length – 4 Bytes – Integer
* Message Elements – (size = Message Elements Length) – type Char \*

Configuration Update Message sent by WUM has a specific structure. This message contains different elements of IEEE 802.11 bindings. We have implemented Channel and txpower bindings.

### Channel message element

* Type: OFDM control element type = 1033 (refer to RFC 5416) – size 4 bytes – type int
* Length : Length of message element – size 4 bytes – type int
* RadioID : radio ID of WTP – size 1 byte – type char
* Reserved: 1 byte of reserved space
* Current Channel: Channel value – Size 1 byte – type char (converted to ASCII value to integer)
* Band Support: size 1 byte – type char
* TI Threshold: size 4 bytes – type int

### TxPower message element

* Type: TxPower element type = 1041 (refer to RFC 5416) – size 4 bytes – type int
* Length : Length of message element – size 4 bytes – type int
* RadioID : radio ID of WTP – size 1 byte – type char
* Reserved: 1 byte of reserved space
* Current TxPower: TxPower value – Size 2 byte – type short (power should be in mW)

More bindings can be added similar to channel and txpower.

## WTP Data retrieval Application

AC runs an external application for gathering data from different WTPs. WTPs send radio resource management information to AC in a CAPWAP data message.

To distinguish this CAPWAP data message from other data messages, we set the value of dataRate in the transport header to 255. When AC receives this data message, it identifies this message as DATA\_STATS and sends it to a UNIX SOCKET.

AC creates a file descriptor for each WTP connected. This is defined in file ACRunState.c in function ACEnterRun(). The WTP Data Retrieval (WDR) Application runs as a separate process for each WTP on the AC. AC forwards the DATA\_STATS message to WDR. WDR then parses the information in this message and updates the information stored in the database.

Presently, only station information is retrieved from the WTP. Station information is received from WTP side in the STA\_INFO struct (ACDataInfoClient.h).