RTLS

Integrating with the RTLS data feed

Aug 2012

ARUBA networks

Table of Contents

Overview

The RTLS data feed from the Aruba controller is designed to send information about client devices that are heard by the Aruba network so that they can be located. This guide outlines how to integrate 3rd party applications with the RTLS data feed. The guide is based on 6.1.3.5-AirGroup.

Aruba offers a few different solutions for location tracking wireless devices associated to the Aruba network. The controller has an XML API built into it that is part of RFLocate. The controller can be polled for the location of a single client at a time. This API is not recommended for location tracking integration. It does not scale to frequent and multiple requests to a large number of devices and may have significant lag.

AirWave's VisualRF also has some XML APIs that can be used to locate clients associated to the wireless network. It was designed with network management in mind and has some limitations for RTLS applications. It locates clients with an accuracy of roughly 5-7 meters of accuracy with about 10-15 minutes of lag.

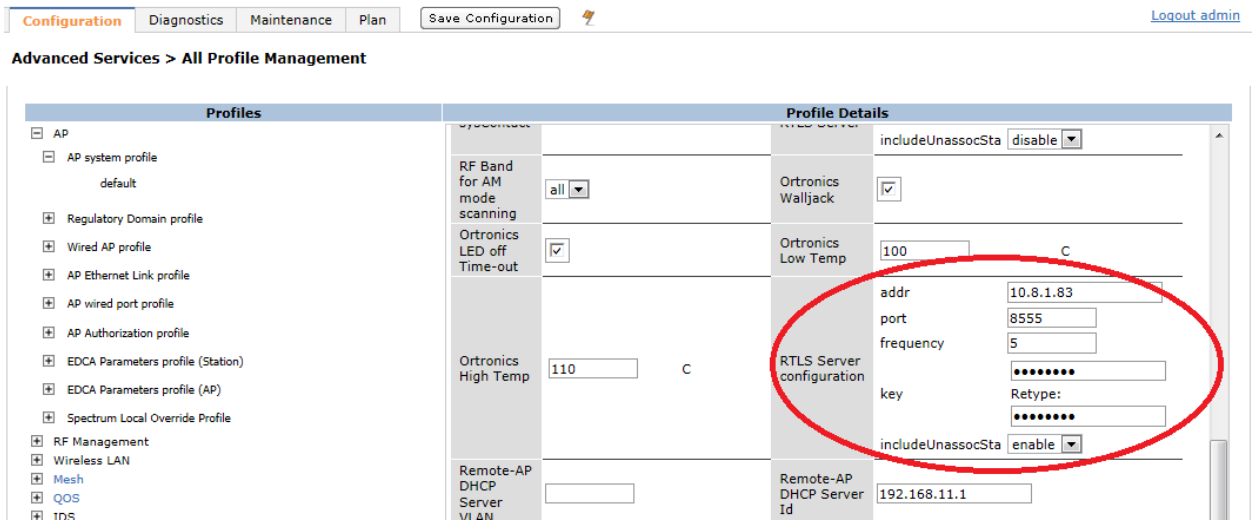
The RTLS location data feed in the controller is designed to enable partner RTLS applications. In AOS 6.1.3.4-AirGroup, the RTLS feed will send info for associated. Prior versions of AOS only send data about clients that are associated to any APs within earshot of the Aruba AP. The unassociated client data is based on probe request information. The associated client data is based on management and data frames excluding probe requests.

Configuring the Controller

Navigate to the AP System profile and find the RTLS settings. Please note that there are two different RTLS server configuration sections. There is the standard feed, RTLS Server Configuration, and an Aeroscout specific feed, AeroScout RTLS. The standard feed is what should be used by 3rd party partners.

Set the IP address, port and key to be used for the data feed. The IP address is the address of the location server. The port is the port used for communication between the controller and the location server. The key should match the value set on the location server. It is used to sign the packets to ensure their validity. The update frequency specifies how frequently updates should be sent for a client and is measured in seconds. The default is 30 seconds. With a 30 second default, the AP will send an update every second with 1/30th of the client devices. The client devices will be spread out across the 30 seconds based on MAC address. There will be an update every 30 seconds for each client. Increasing the frequency can have a negative impact on location data in congested wireless networks. The includeUnassocSta flag will cause the unassociated client device data to be included in the feed. In this case, unassociated clients mean devices that are not associated to any AP.

Please note that the traffic will be sent UDP.



Additional details can be found in the AOS user guide which can be downloaded from support.arubanetworks.com.

Initiating the feed

Once the server information is entered into the controller, the controller will configure the AP group to send data to the RTLS server. The AP will start sending AR_AS_notifications every 10 seconds as a UDP packets until it receives an acknowledgment. The ack should be in the form of an AR_ACK. The AR_ACK should be the RTLS header with the same message ID as the notification. The payload should be empty. That header will then need to be signed. Use the entire packet and the shared secret to create the signature that is then appended to the end of the packet.

Once the AR_ACK makes it back to the AP, Both the tag and station reporting will be enabled. This can be verified by monitoring the new traffic coming from the AP. Or by running a debug command on the controller. Run 'show ap monitor debug status ap-name <ap_name>' to view details about the AP. Look at the RTLS Configuration and State section of the output. You should see a row for RTLS and the Active column should be checked.

Packet Format Details

Following is a brief description of RTLS protocol.

Standard Message Format

```
-------------------------------------------------------
| ip | udp | rtls_hdr | rtls_payload | rtls_signature |
-------------------------------------------------------
```

RTLS Header

Field Name	Bytes	Notes
Message Type	2	
Message Id	2	Typically increasing
Major version	1	Set to 1 or 2
Minor Version	1	Set to 0
Data Length	2	Length of RTLS payload only
AP MAC	6	AP MAC. Unique identifier for an AP
Padding	2	

RTLS Signature

A 20 byte signature is included at the end of every message. This is a HMAC-SHA1 signature created by using the shared secret as the key and the contents of RTLS packet as the data.

RTLS Payload

Payload is the optional payload that may be present in tag transmissions. Payload will not be present for clients or unassociated clients.

Message Codes

Message Type	Code
AR_AS_CONFIG_SET	0x0000
AR_STATION_REQUEST	0x0001
AR_ACK	0x0010
AR_NACK	0x0011
AR_TAG_REPORT	0x0012
AR_STATION_REPORT	0x0013
AR_COMPOUND_MESSAGE_REPORT	0x0014
AR_AP_NOTIFICATION	0x0015
AR_MMS_CONFIG_SET	0x0016
AR_STATION_EX_REPORT	0x0017
AR_AP_EX_REPORT	0x0018

AR_AS_CONFIG_SET

This message is used with the Aeroscout RTLS feed and should does not affect the standard feed. Tis message will set the RTLS detected Multicast Address.

010ccc000000 is the default in the AP. If this message is sent, it will overwrite the hardcoded value.

Field Name	Bytes	Notes
As_tag_addr	6	Tag multicast address
Reserved	2	

On reception of this message, AP will generate an AR_ACK with the same message id as the AR_AS_CONFIG_SET message.

AR_STATION_REQUEST

This message can be sent to the AP from RTLS server to get latest RSSI for a client or an AP. AP will generate a AR_STATION_RESPONSE message as a response to the request. If station is not found, AP will generate a NACK with error-code, AR_NACK_STATION_NOT_FOUND.

Field Name	Bytes	Notes
MAC address	6	
Reserved	2	

AR_ACK

This is generated by receiver for some messages. Id in the ACK is the same as the id in the message being ACKed. ACK message has no payload.

AR_NACK

This is generated by receiver for some messages. id in the NACK is the same as the id in the message being NACKed.

```
#define AR_NACK_INTERNAL_ERROR    0
#define AR_NACK_STATION_NOT_FOUND 1

   __u16 flags; // flags as defined above
   __u8  reserved[2];
```

AR_TAG_REPORT

This is generated by AP when a chirp frame is seen in the air. RTLS server will NOT generate an ACK on receipt of this message. The chirp should use 01:18:8e:00:00:00 as the multicast address and have the to and from ds bits set to 1 (true).

Field	Bytes	Description	Notes
Bssid	6	MAC address of the radio where the frame was received	

RSSI	1	Signal as a signed negative hex value	Convert Hex to decimal and subtract 256 to get the signal value.
Noise_floor	1	Noise floor of the radio	
Timestamp	4	Millisecond granularity timestamp that represents local time in AP when message was sent. Timestamps across multiple APs are not synchronized.	
Tag_mac	6	MAC address of the tag	
Frame_control	2	Frame control from 802.11 header	
Sequence	2	Sequence number from the 802.11 header	
Data rate	1	Data rate of chirp frame	1 = 0x00 2 = 0x01 5.5 = 0x02 6 = 0x03 9 = 0x04 11 = 0x05 12 = 0x06 18 = 0x07 24 = 0x08 36 = 0x09 48 = 0x0A 54 = 0x0B
Tx_power	1	Transmit power in dbm.	0xff = not available
Channel	1	Channel of tag transmission	
Battery	1	Batter level information from the chirp frame if present.	0xff = not available
Reserved	2		
Payload		Chirp frame payload if any	__u8 payload[0]; // chirp frame payload, if any

AR_STATION_REPORT

This is generated as a response to AR_STATION_REQUEST or periodically for a station from the AP. The RTLS server should not generate an ACK on receipt of this message

AR_STATION_REPORT

Field	Bytes	Description	Notes
MAC	6	MAC address of station	Set to BSSID if type=AR_WLAN_AP Set to MAC of station if type=AR_WLAN_CLIENT
Noise floor	1	Noise floor of the channel where the station was last heard	
Data rate	1	Data rate of last transmission	Set to 0 for unassociated stations. Values 1 = 0x00 2 = 0x01 5.5 = 0x02 6 = 0x03 9 = 0x04 11 = 0x05 12 = 0x06 18 = 0x07 24 = 0x08 36 = 0x09 48 = 0x0A 54 = 0x0B
Channel	1	Channel where station was last heard	For unassociated stations this may change a lot
RSSI	1	Signal as a signed negative hex value	Convert Hex to decimal and subtract 256 to get the signal value.
Type	1	Type of device	1 = AR_WLAN_CLIENT, 2 = AR_WLAN_AP
Associated	1	Association status of station	1 = AR_WLAN_ASSOCIATED (All APs and Associated Stations), 2 = AR_WLAN_UNASSOCIATED (Unassociated Stations)
Radio_BSSID	6	BSSID of the radio that	

		detected the device	
Mon_BSSID	6	BSSID of the AP that the station is associated to	Will be set to 0s for unassociated stations
Age	4	The number of seconds since the last packet was heard from this station.	

AR_STATION_EX_REPORT

This message is typically sent as contained in AR_COMPOUND_MESSAGE_REPORT.

Field	Bytes	Description	Notes
MAC	6	MAC address of station	
BSSID	6	BSSID with which this station is associated	
ESSID	33	ESSID with which this station is associated	
Channel	1	Channel where this station is active	
Phy_type	1	1 = 802.11b 2= 80.11a 3=802.11g 4=802.11ag	
RSSI	1	Average RSSI during the duration. RSSI is signal strength. Signal is a signed negative hex value.	Convert Hex to decimal and subtract 256 to get the signal value.
Duration	2	Average calculation duration	
Num_packets	2	Number of packets used in average RSSI calculation	
Noise_floor	1	Noise floor of the radio	
Classification	1	1 = valid 2 = interfering 3 = DOS'ed	
Reserved	2	reserved	

AR_AP_EX_REPORT

This message is typically sent as contained in AR_COMPOUND_MESSAGE_REPORT.

Field	Bytes	Description	Notes
BSSID	6	BSSID with which this station is associated	

ESSID	33	ESSID with which this station is associated	
Channel	1	Channel where this station is active	
Phy_type	1	1 = 802.11b 2= 80.11a 3=802.11g 4=802.11ag	
RSSI	1	Average RSSI during the duration. RSSI is signal strength. Signal is a signed negative hex value.	Convert Hex to decimal and subtract 256 to get the signal value.
Duration	2	Average calculation duration	
Num_packets	2	Number of packets used in average RSSI calculation	
Noise_floor	1	Noise floor of the radio	
Classification	1	1 = valid 2 = interfering 3 = DOS'ed	
Match_type	1	Internal Aruba use	
Match_method	1	Internal Aruba use	
Reserved	2	reserved	

AR_COMPOUND_MESSAGE_REPORT

This message contains a series of AR_TAG_REPORT, AR_STATION_REPORT, AR_STATION_EX_REPORT and AR_AP_EX_REPORT. Typically it consists of AR_STATION_EX_REPORT AND AR_AP_EX_REPORT.

```
__u16 messages; // number of messages, not accurate at this point
__u8  reserved[2];

// other message types follow, typically AR_STATION_EX_REPORT and
AR_AP_EX_REPORT
__u8  payload[0];
```

General Questions

How frequently are updates sent?

This question is tricky since it isn't specific enough.

The update interval sets how often updated information for a single device will be sent in the RTLS data stream. If it is set to 30 seconds, rtls data for a specific client will be sent every 30 seconds.

But if the question really means 'how often should the RTLS server expect an update from the controller' there is a different answer. The controller attempts to smooth out the data flowing to the server to avoid large spikes in calculation on the server side. The controller will send an update as frequently as every second if there is data to send. Which client has data sent is determined by a MAC address hash. This may result in data that is not completely smooth. Some updates may have a few more clients than others, but the overall effect is that there will be frequent small updates rather than a single large one.

Imagine the update interval is set to 30 seconds. If there 30 clients, there will be an update roughly every second that includes a single clients data. Now if there are 90 clients heard, there will still be an update every second but it will include data for 3 clients. If there are only 10 clients, there will be some seconds where no updates are sent. There will be 10 updates in that 30 seconds.

How much bandwidth will be used by RTLS?

This is a tough question to answer without knowing the exact number of clients that will be heard. Every payload will have a header and a signature which are 16 and 20 bytes respectively. Then the number of clients included will make a difference. A single client AR_Station_Report will be 28 bytes. Next you need to know the update interval. For this example we will use the default of 30 seconds. See previous question for details on update intervals.

The formulae to use to calculate the approximate bytes per second is (header size)+(signature size)+(1/update interval)*(num of clients)*(client data size).

If we assume there are 90 clients being heard, then the average Bps will be (16)+(20)+(1/30)(90)(28) = 120 Bytes per second for a single AP.

Sample Code:

This code is show as an example of how to parse the compound message reports and how to sign the packets. It makes references to utility functions that are not defined here. It is not code that can be directly reused as it will not work without those utility functions.

```
sub decode_compound_message_report {
  my ($self, $payload, $opaque_hr) = @_;
```

```
# The number of messages is included but not accurate currently.  Skip the first
two bytes of padding.
  my $compound = substr($payload, 4);

  # Now decode the encapsulated payloads
  my $i = 0;
  while ($i < length($compound)) {
    my $len = $self->_decode(substr($compound, $i), $opaque_hr);
    $i += $len;
  }
}
sub _decode {
  my ($self, $buf, $opaquen_hr) = @_;

  my %header = $self->header_of($buf);
  return unless $header{major_version}; # heuristic for proper decode
  my $payload = substr($buf, AR_HEADER_LEN, $header{data_len});

  if ($payload) {
    $self->dispatch_decode($header{code}, $payload, {
      ap_mac => $header{ap_mac},
      %$opaque_hr,
    });
  }

  # handle no-payload packet types here
  if ($header{code} == AR_AP_NOTIFICATION) {
    $self->{cb_ap_notification}->({
      ap_mac => $header{ap_mac},
      id => $header{id},
      %$opaque_hr,
    });
  } elsif ($header{code} == AR_ACK) {
    $self->{cb_ap_ack}->({
      ap_mac => $header{ap_mac},
      id => $header{id},
      %$opaque_hr,
    });
  }

  return (AR_HEADER_LEN + $header{data_len});
```

```
}

sub dispatch_decode {
  my ($self, $code, $packet, $opaque_hr) = @_;

  if ($code == AR_TAG_REPORT) {
    $self->decode_tag_report($packet, $opaque_hr);
  } elsif ($code == AR_STATION_REPORT) {
    $self->decode_station_report($packet, $opaque_hr);
  } elsif ($code == AR_COMPOUND_MESSAGE_REPORT) {
    $self->decode_compound_message_report($packet, $opaque_hr);
  } elsif ($code == AR_AP_NOTIFICATION) {
    $self->decode_ap_notification($packet, $opaque_hr);
  } elsif ($code == AR_STATION_EX_REPORT) {
    $self->decode_station_ex_report($packet, $opaque_hr);
  } elsif ($code == AR_AP_EX_REPORT) {
    $self->decode_ap_ex_report($packet, $opaque_hr);
  } elsif ($code == AR_NACK) {
    $self->decode_ap_nack($packet, $opaque_hr);
  }
}

sub decode {
  my ($self, $buf, $opaque_hr) = @_;
  $opaque_hr ||= {}; # it's optional

  # validate signature
  my $actual_sig = substr($buf, - AR_SIGNATURE_LEN);
  my $expected_sig = $self->signature_of_packet($buf);
  return unless ($actual_sig eq $expected_sig);

  $self->_decode($buf, $opaque_hr);
}
sub encode {
  my ($self, %rtls_data) = @_;

  unless (exists $rtls_data{id}) {
    $rtls_data{id} = $self->{_id};
    $self->{_id} = ($self->{_id} + 1) % 2**16;
  }
  $rtls_data{payload} ||= '';
```

```
  $rtls_data{major_version} ||= 1;
  $rtls_data{minor_version} ||= ($self->{mms} ? 1 : 0);
  my $packet = pack($self->header_pack_str,
    $rtls_data{code},
    $rtls_data{id},
    $rtls_data{major_version},
    $rtls_data{minor_version},
    length($rtls_data{payload}),
    Mercury::Utility::MAC->text2bin($rtls_data{target_mac}),
    "\0\0",
  );
  $packet .= $rtls_data{payload};
  $packet .= $self->signature_of($packet);
  return $packet;
}
sub signature_of {
  my ($self, $data) = @_;

  my $key;
  $key = $self->{password};
    return hmac_sha1($data, $key);
}
```

About Aruba Networks

Aruba is the global leader in distributed enterprise networks. Its award-winning portfolio of campus, branch/teleworker, and mobile solutions simplify operations and secure access to all corporate applications and services - regardless of the user's device, location, or network. This dramatically improves productivity and lowers capital and operational costs.

Listed on the NASDAQ and Russell 2000® Index, Aruba is based in Sunnyvale, California, and has operations throughout the Americas, Europe, Middle East, and Asia Pacific regions. To learn more, visit Aruba at http://www.arubanetworks.com. For real-time news updates follow Aruba on Twitter, Facebook, or the Green Island News Blog.

1344 Crossman Ave. Sunnyvale, CA 94089-1113
Tel.. 1.408.227.4500 | Fax. +1.408.227.4550 | info@arubanetworks.com | arubanetworks.com