Wyres

LoWAPP Functional specification

2016

Brian Wyld, Nathan Olff
21/11/2016

This document

Objectives

This document presents the functional specification and technical details for the “LoRa Wide Area Peer Protocol” (LoWAPP), a small scale communication protocol between LoRa based peer endpoints. Sufficient information is given to enable the implementation of a conformant node in such a network.

Structure

The document presents an overview of the requirements of the system, a detailed architecture proposition to address them, the packet level formats and the state machines required. Finally, a future work section indicates open questions and future directions.

Versions

Version	Author	Date	Changes
V1.0	B.Wyld	01/01/2016	Creation
V1.1	B.Wyld	04/02/2016	Addition of architecture diagrams
V1.2	B.Wyld	18/02/2016	Addition of AT command set, addition of groupId, add state machine
V1.3	B.Wyld	20/02/2016	Addition of licensing agreement
V1.4	B.Wyld	02/03/2016	Updated with comments from NS
V1.5	B.Wyld	08/04/2016	Name change
V1.6	B.Wyld	13/05/2016	Updated based on implementation start
V1.7	B.Wyld	01/06/2016	Updated based on brainstorming and comments from EL
V1.8	N.Olff	21/11/2016	Updated based on simulation results and implementation

License

LoWAPP

Protocol requirements

- Groups of endpoint devices with up to 250 members
- Identification of each group via optional 2 byte ‘groupId’
- Explicit identification of each device in the group
- Encryption of messages within the group
- Explicit ack of messages sent to a specific device
- Rejection of duplicate messages from specific devices (to specific or broadcast)
- Messages of 1-246 bytes for end to end or broadcast communication, and 1-246 bytes for external communication using gateway nodes
- Low power requirements for reception
- No ‘hard’ clock sync required between nodes
- No requirement to have a nominated/dedicated node for message exchanges within the group
- Use of low cost low power long range radio chip (Sx1272)
- Explicit configuration by application layer per node (no discovery or configuration negotiation in the protocol)

Overview

LoWAPP is a LoWPAN type protocol, similar to Zigbee or Thread, but focused on providing the peer-peer interactions rather than generic mesh routing functionality up to some central server. Unlike these protocols, it does not require any ‘special’ function nodes to communicate between peers, nor does it require any nodes that are permanently awake and therefore powered.

The protocol proposes a MAC layer that uses a specific mode of the LoRa radio physical layer. The MAC layer provides a means for 2-250 endpoint nodes (endpoint implies using chipsets that can access a single radio channel/Spreading Factor (SF) at a time) to exchange datagram messages, either explicitly addressed or broadcast to the group. Messages are encrypted with an AES128 key, which is a shared secret between the nodes of the group. The MAC provides for acknowledgement of messages and the detection of lost and duplicated messages.

Also proposed is an ad-hoc network routing layer which provides a means to extend the reach outside of the peer group. This is done by designation of an optional ‘gateway’ node, which allows messages to be exchanged with other groups (identified by an application level allocated ‘groupId’ or ‘Internet’ level hosts). The actual addressing of these hosts is controlled by the per-node application, not automatically by the network layer.

The group is explicitly configured for each node. No dynamic means for discovery or joining is provided within the protocol. Each node must be configured (via an external means outside the scope of the protocol, typically either static manual allocation or application level process) with:

- a unique device id between 1 and 250
- the shared encryption key (128 bits AES)
- a specific radio channel and SF to use for all rx/tx exchanges

Note :

- Any node with the key can decrypt all messages, whether addressed to their device id or not. Hence, ‘cracking’ any node member of the group and obtaining the key breaks the security of the entire group.
- No mechanism is provided to ensure device id uniqueness – the provisioning agent is responsible for determining a mechanism to avoid reuse.

Any node can:

- Receive a message and determine if they are intended for their id, or for the broadcast id
- Send a message to a specific node id or the broadcast
- Receive an explicit ack for specifically addressed messages
- Validate if a ‘external gateway’ Is available and use it to:
    - address messages to it for forwarding to any public IPv4 address
    - Receive messages from the gateway (received for a specific node id or the broadcast id)

Latency to receive a message is between 0.1 and 1.1s.

Power requirements are governed by a rx scanning duty cycle around 1% (when using a symbol time of 1ms for a preamble time of 1 second) to reliably receive messages and a transmission requirement of 1.1s per packet transmitted (these values use SF7)

Note that use of the LoRa radio channel and the specific power requirements mean that the protocol is intended for infrequent message exchanges of size <256 bytes (including MAC header), where low energy devices will spend most of their time sleeping and periodically scanning for incoming messages rather than transmitting (since transmission is much more costly than reception)

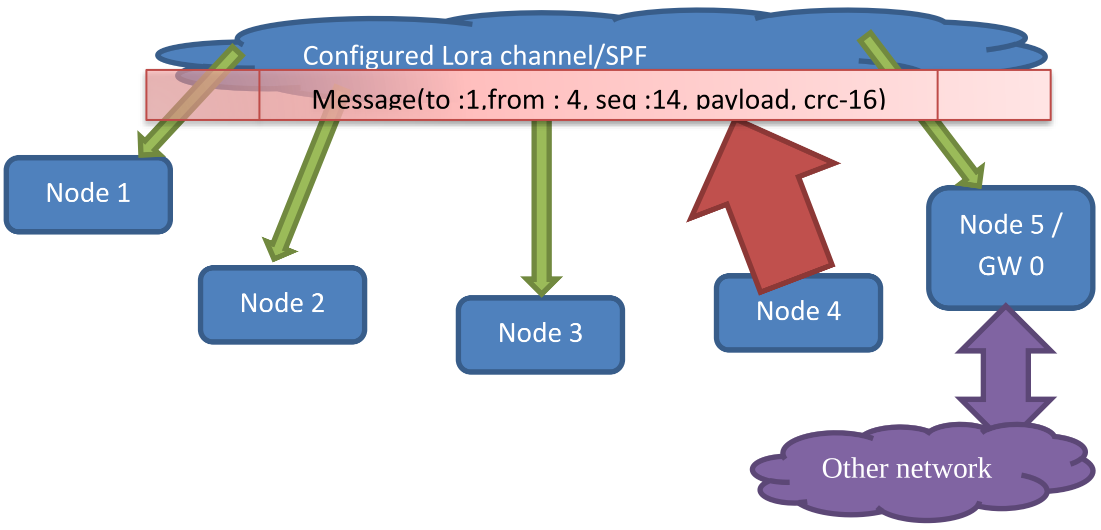
Architecture / operation

Overview

There are 3 main elements to the operation to achieve the desired requirements. These are:

- A basic node2node acknowledged and sequenced datagram protocol. Every tx to a specific id is acknowledged. Each device stores a group of sequence numbers for each possible device id in the group. Sequence numbers allow rejection of duplicate frames and detection of missed ones. Ack failures or missing frame recovery is left to the application layer.
- A gateway profile and explicit message header, which allows a node to designate itself as the forwarding gateway to explicitly addressed protocols / devices such as internet/ipV4 or LoRaWAN/Device EUI. Two way exchanges are possible by the gateway mapping external received messages to local device ids using a protocol dependent mechanism (a proposition for IPV4/UDP is given later)
- Use of the 'CAD' (Channel Activity Detection) mode of the LoRa radio chip to achieve low (ish) message latency with low power duty cycles without costly centralized clock sync systems. This involves the radio waking up to listen for one symbol every second for the pre-amble (this equates to 1ms with SF7, 32ms with SF12) when receiving, and using a preamble of (for example) 1s for every transmission. SF7 thus gives a 'rx scan duty cycle' of just 0.1%... This duty cycle ratio can be adjusted to suit specific cases (length of the preamble in ms being a configuration parameter for the group)

The group architecture is shown below with a message being transmitted from node 4 to node 1.


Configured Lora channel/SPF
Message(to :1,from : 4, seq :14, payload, crc-16)
Node 1
Node 2
Node 3
Node 4
Node 5 / GW 0
Other network

Operation

Node configuration

Each node is pre-configured with

- A 2 byte groupId, by default 0x0000, but which can be assigned to allow routing between disjoint groups (via a gateway node)
- A unique-in-group 1 byte device id, between 1 and 250
- A 128 bits AES key
- The LoRa radio channel id to use (between 0 and 15, each channel id corresponding to a radio frequency)
- The LoRa SF to use, between 7 and 12 (default 7 -> 1ms CAD scan time)
- The preamble length in ms (default 1000) : this fixes the 'receive check' wakeup period and the overall power requirement.

Device ids:

6 specific device ids are reserved:

- 0xFF : broadcast : a message sent to this id is intended for all nodes in the group. It will not be acknowledged.
- 0x00 : gateway outbound request. The payload of the message is specifically defined for messages to this id. An ack means that a gateway node accepted to forward the message.
- 0xFE/0xFD/0xFC/0xFB : reserved for future us

Sequence numbers

Each node keeps two tx specific and one rx specific 1 byte sequence numbers for each possible destination node including the broadcast/gateway (256 values). The values start at 0 when the node is reset, and loop at 255 round to 1 (hence 0 is only ever seen if the remote node has reset). One sequence number (called out_txseq) is used to keep track of the messages sent to a node. It is sent in the basic node2node messages. Another one (called out_rxseq) is used to store the last sequence number received in an acknowledgement from a node. A third one (called in_expected) stores the last sequence number received in a standard message from a specific node.

Missing reception is assumed if the received value is greater (mod 256) than the expected value (by delta<127). Duplicate frames is assumed if it is less (mod 256) than expected (by delta <10). Duplicate frames are discarded, missing frame events are signaled up to the application layer.

Two of those sequence numbers are sent in the acknowledgement messages so that the receiver of an acknowledgement can detect if it missed a previously sent acknowledgement, and therefore mistakenly notified the application of a failed transmission.

Message transmission

To transmit a message, the payload of 1-246 bytes is prefixed with the LoWAPP header (see later), suffixed with a CRC16 and encrypted (see later). The entire message is then sent to the radio for transmission with a preamble of X ms (as configured) on the specific channel/SF, in LBT (Listen Before Talk) mode. Once the last bit is transmitted, the out_txseq sequence number is incremented.

One second after the transmission finished, the transmitter goes into reception mode on the same channel/SF. It then listens for up to one second to receive the acknowledgement (which has a 'classic' preamble of 8 symbols). This time slot is done to allow the receiver to decode the message before acknowledging it.

The ack message contains the updated sequence number for the receiver.

If the ack is received correctly, then the out_rxseq sequence number for this destination is incremented for the next message.

If the ack is not received correctly or does not match the expected sequence numbers, the application is notified of the missing frame of missing acknowledgement from the difference of sequence numbers. If no ack is received, the application is also informed.

Message reception

CAD mode is used to listen intermittently for the long preamble. In between the radio is in sleep mode and the CPU can sleep or deal with other actions.

Note : see the C.A.D. section in the SX1272 app notes for the details

Once the message payload is received, the node performs the AES decryption and checks the CRC-16. If the CRC does not match, the packet is discarded (assumed to belong to another group).

If the destination id is this specific node, the packet sequence number is checked and compare to the in_expected sequence number. If they are equal, the packet is sent to the application and the in_expected sequence number of incremented (%255+1).

The acknowledgement is only sent after a period of time equivalent to the middle of the ack time slot (see above). This corresponds to about 1.5s after reception.

Unexpected sequence numbers either silently discard the packet or generate a missing packet event to the application. In both cases an ack with the expected sequence number and received sequence number is sent.

In all cases a 'silent' time is observed after packet reception to avoid a clash with the ack packet that the destination is assumed to be transmitting.

Encryption/security

The AES128 key is used to encrypt/decrypt each packet. This requires a shared 'nonce' between encoder/decoder which must evolve in time.

The nonce used in LoWAPP is the concatenation of the groupId (known by all participants in the group), and a pseudo random 2 byte value transmitted at the start of every packet outside the CRC/encrypted segment. The transmitter of the packet can generate this value by any means.

The actual key used to encrypt the payload is the output of a XOR operation (exclusive or logic operation) between the encryption key and the nonce (used 4 times to match the key size).

TBD is this best method? Alternative : only encrypt payload, generate a MIC to secure the overall message, but have control fields in clear. Can get rid of CRC, nonce, and discard rx frames immediately if destId !=us. TBD pros/cons.

Packet formats

Note that the Lora PHY header is used for all packets, and includes the overall packet length.

In all packets the block following the 'nonce' field has been encrypted.

Lora hdr (included in all LoWAPP messages)

Version (4-bits)	Message type (4-bits)	payloadLength (1 byte)	RFU (2 bytes)

Basic peer2peer

Preamble (8 sym)	Lora hdr (4)	Nonce (2)	destId (1)	srcId (1)	txSeq (1)	payload (1-246)	Crc16 (2)

Ack

Preamble (8 sym)	Lora hdr (4)	Nonce (2)	destId (1)	srcId (1)	rxdSeq (1)	expectedSeq (1)	Crc16 (2)

Broadcast

Preamble (8 sym)	Lora hdr (4)	Nonce (2)	0xFF (1)	srcId (1)	txSeq (1)	payload (1-246)	Crc16 (2)

Gateway outbound

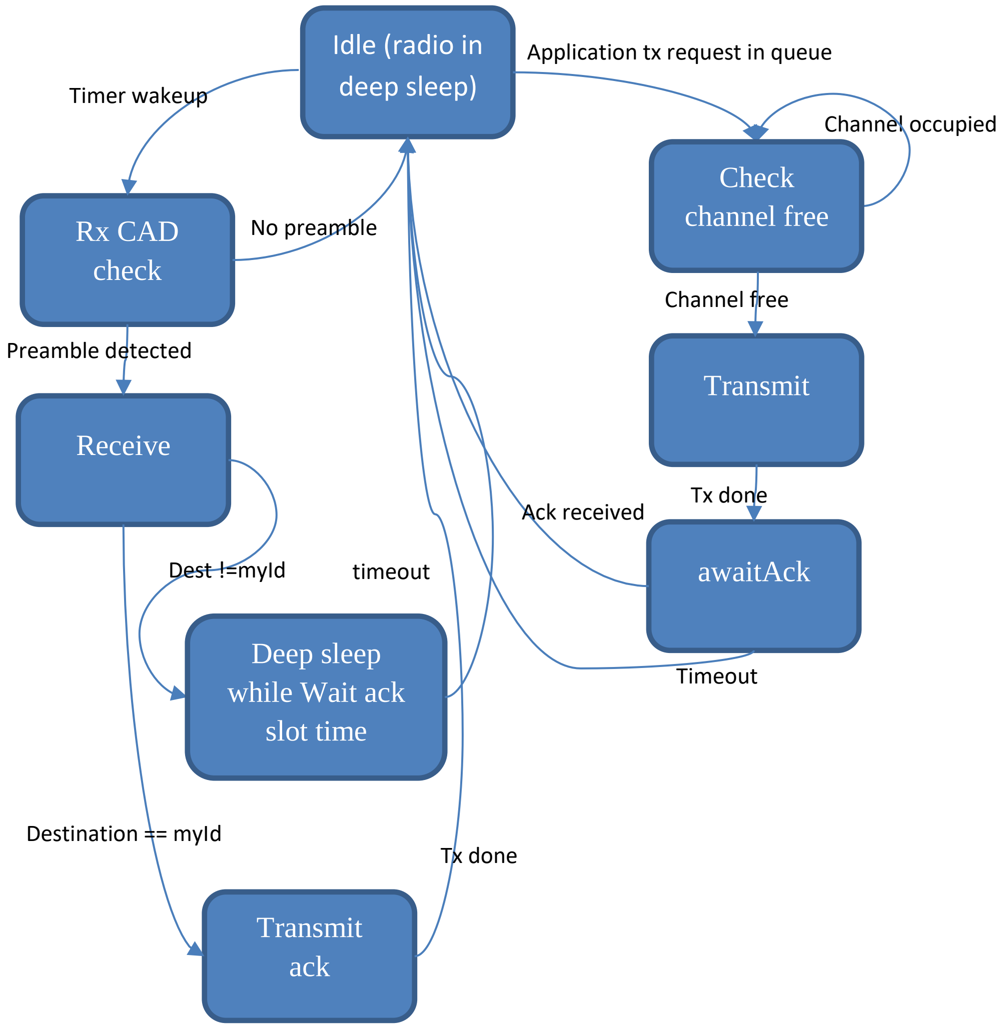
Preamble (1000ms)	Lora hdr (4)	Nonce (2)	0x00 (1)	srcId (1)	Tx seqNb (1)	netType (4)	netAddrLen (1)	netAddr (N)	Payload (1-226)	Crc16 (2)

Gateway inbound

Preamble (1000ms)	Lora hdr (4)	Nonce (2)	destId (1)	0x00 (1)	Tx seqNb (1)	netType (4)	netAddrLen (1)	netAddr (N)	Payload (1-226)	Crc16 (2)

State machines

The explicit state required in the protocol is the switch between reception mode (when preamble is detected), transmission/ack reception and sleep. This is shown in the diagram below.



Notes:

- The ack transmission is in a reserved timeslot following the reception of a packet : only the explicit destination node may transmit within this slot (all nodes should have detected the packet and so will wait and not tx during the slot).
- The treatment of the sequence numbers does not require explicit state.

Gateway operation

A node designates itself as a gateway by application level configuration.

A gateway is defined as a node that has a ‘forwarding’ means of communication, to another LoWAPP group or to another communication network.

Gateway nodes use a 32-bit bitmask to specify the type of messages it can forward.

Currently defined types:

0x00000001 : LoWAPP group : ie the node has 2 or more group configurations with unique groupId values

0x00000002 : LoRaWAN public network : ie the node has a LoRaWAN network configuration

0x00000004 : IPv4 network with UDP addressing : ie the node has an IPv4 connection

0x00000008 : CoAP network request : ie the node has a IPv4/v6 connection

Each network type has a specific networking address block which allows the forwarding mechanism to address a specific node in the remote network. The definition of these blocks is shown below:

LoWAPP : destination deviceId in remote group, remote group groupId

LoRaWAN : destination AppEUI

IPv4/UDP : destination IPv4/remote UDP port. The local UDP port is assigned as 0x38xx where xx is the local deviceId.

CoAP : TBD

AT Command set

A LoRa node implementing this protocol will typically act as a communication channel for the application. The control of the node is generally done via a message oriented serial channel eg UART. Classically, such ‘modem’ nodes are controlled via a command set historically based on ‘AT’ commands. The same command/response exchange can operate in a tightly coupled environment via message passing primitives.

The operation can be either ‘pull’ (the application must explicitly issue an AT command to receive a response from the card) or ‘push’ (the application may issue AT commands and receive responses, it may also receive unsolicited data). Generally, the card initializes in the ‘pull’ mode, and is switched into push at the behest of the application.

The following tables define the various commands, divided into configuration, operation and validation type commands.

Command (no parameters) format : AT+<cmd>

Command (parameters) format : AT+<cmd>=<value>{,<value>}   // optional space round = and ,

Response : OK|NOK { <json object> }   // note obligatory space between OK/NOK and the optional json

Configuration

Configuration is always of a string key and a string value (even if this is then parsed as a decimal or Boolean). Keys are case-insensitive ie groupId==GROUPID. On output they are always emitted as lower case.

Get config value associated with <key> : AT+ATTR

Set config value associated with <key> : AT+ATTR=<value>

Command	**Value**	**Action**	**Response**
AT+ GROUPID=<value>	4 hex digits giving the id for this group (default 0000)	Set groupId	OK
AT+GROUPID	None	Get groupId	OK { “groupId”:”<The 4 hex digits of the groupId>” }
AT+DEVICEID=<value>	2 hex digits giving the device id (default 01)	Set deviceId	OK
AT+DEVICEID	None	Get deviceId	OK { “deviceid”:”<The 2 hex digits of the deviceid>” }
AT+GWMASK=<value>	8 hex digits for the bitmask of gateway functionalities	Enable/disable the processing of gateway messages	OK
AT+GWMASK	None	Get gateway flag	OK { “gwMask”: <The 8 hex digits of the gateway mask>}

AT+ENCKEY=<value>	16 hex digits giving the 128bit AES key (no default)	Set encoding key	OK
AT+CHANID=<value>	2 hex digits between 00 and 0F, indicating the id of the Lora (LoWAPP specific) to use (default 0=863,125MHz)	Set LoRa channel	OK
AT+CHANID	None	Get LoRa channel	OK { “chanid”: “<2 hex digit chanId>” }
AT+TXDR=<value>	2 hex digits between 07 and 0C indicating the spreading factor to use (7 to 12)	Set LoRa SF	OK
AT+TXDR	None	Get LoRa SF	OK { “sf”: “<2 hex digit SF>” }
AT+PTIME=<value>	Decimal value giving the preamble length to use in ms (default 500)	Set preamble length	OK
AT+PTIME	None	Get preamble length	OK { “pTime”: “<decimal preamble length in ms>” }
AT&W	None	write current configuration to persistent store.	OK if write ok, NOK otherwise

Validation

Command	Parameters	Action	Response
AT&V	None	Print current config	OK { json Object with currently configured values (note that the enckey cannot be displayed) }
AT+SELFTEST	None	Perform hardware test	OK
AT+STATS	None	Get rx/tx statistics	OK { Object of stats in JSON format }
AT+WHO	None	Get list of recently seen group members and their last known RSSIs	OK { “wholist”:[{ “device”:”deviceid”, “lastSeen”:”ts”, “lastRSSI”:”value” },] }
AT+PING=<value>	2 hex digits giving the device id to ping	Send ping packet to the specified member, and block waiting for the ack	OK TX if ack received, NOK TX if not
AT+HELLO	None	Send a hello packet. Hello responses will be received as and when remote nodes decide to reply	OK

Operation

Command	Parameters	Action	Response
AT+SEND=<value>,<value>	2 hex digits giving destination device id, 2*N hex digits giving the payload of N bytes	Send data to the given id	OK if ack received for destination!= broadcast NOK if no ack for non-broadcast NOK if no AES key configured.
AT+POLLRX	None	Check for received packets. Returns all valid received packets since last check.	OK { “rxpkts”:[{ json pkt }xN] }
AT+PUSHRX	None	Change to PUSH mode. The next AT command of any type *except* AT+SEND will change back to PULL mode.	OK PUSHRX Each subsequent received packet is sent up immediately as a JSON structure.
AT+DISCONNECT	None	Stop listening for packets at all, never respond to radio. AT+SEND commands will be rejected.	OK DISCONNECT
AT+CONNECT	None	Start listening for packets and allow transmissions	OK CONNECT
ATZ	None	Reset the device	BOOT OK

Implementation

The implementation uses a fixed set of function calls to talk to the specific hardware card to ensure portability. The required calls are passed as function call pointers to the lowapp_init() function. Minimal libc functionality is also required (sprintf, str* etc).

See lowapp implementation document for more details.

Current state of the implementation

At the moment, the gateway functionality described in this document is not implemented. Device can therefore only communicate with other devices in the same group.

Future study / TBDs

- Gateway functionality
- Improve support for duty cycle requirements
- Application level protocol support for auto-configuration mechanisms (eg use IP address of application node, or elements of a LoRaWAN DeviceEUI…)
- Apply streaming decryption during reception, and abort reception as soon as destination id decoded if not for this node
- Reducing awake time via use of 2nd channel for ack only (no need for all nodes to be awake during this time
- Use of multiple channels by listening on N channels one after the other, and transmitting on random channel within the group

Related protocols and reading

PHY/MAC layers for LoWPAN:

802.15.4:
https://en.wikipedia.org/wiki/IEEE_802.15.4

Network and application protocols:

Zigbee
https://en.wikipedia.org/wiki/ZigBee

Why not : , not peer-to-peer, requires use of 802.15.4, member only group, costs at least ?$ per year, not open

Thread
https://en.wikipedia.org/wiki/Thread_(network_protocol)

http://threadgroup.org/Portals/0/documents/whitepapers/Thread%20Stack%20Fundamentals_v2_public.pdf

Why not : , not peer-to-peer, requires use of 802.15.4, member only group, costs at least 2500$ per year, not open

6LoWPAN
RFC6282

Why not : , not peer-to-peer, requires use of IPv6, complex, requires dedicated nodes

IPv4/UDP
Why not : headers too long, IPv4 addresses no longer available, UDP has no acks, orientated for big networks