APRS 438 Protocol

Serge Y. Stroobandt, ON4AA

Aug 08, 2023

CONTENTS

Welcome to the home of **APRS 438**, the 438 MHz amateur radio LoRa automatic packet reporting system that **extends range by saving bytes.**

Unlike some other ham radio LoRa APRS projects, this project aims at **deploying LoRa the way it was intended;** namely by being frugal about the number of bytes put on air. Doing so, reaps a number of benefits:

- less airtime,
- increased battery life,
- higher chances of good packet reception,
- hence, increased range,
- lower probability of packet collisions,
- therefore, more channel capacity.

In dense urban environments and/or on flat terrain, **LoRa works best when the data payload is kept to a strict minimum.** This can be achieved by taking full advantage of all 256 characters available for transmission with LoRa. The APRS frame compression protocols presented in this white paper aim precisely at doing that; for LoRa, *or any other data link with an extended character set.*

ESP32 firmware adhering to these compression protocols is provided as well:

- Terminal documentation
- Tracker documentation
- i-gate documentation

Caution: Unlike the vast majority of other LoRa projects, the firmware of this project employs **licensed frequency spectrum** exclusive to the use of amateur radio. **You need a valid amateur radio license to be able to use APRS 438 firmware.** Contact your national government or local amateur radio club to find out how to obtain an amateur radio license.

Attention: This document is still subject to change. Check regularly for changes and added clarifications.

CHAPTER

ONE

LINK PARAMETERS

The following LoRa link parameters are proposed for amateur radio LoRa APRS 438:

Table 1.1: APRS 438 link parameters

LoRa parameter	uplink	downlink	alternative downlink
frequency	438.050 MHz	439.550 MHz	434.425 MHz
upchirp bandwith BW	125 kHz	125 kHz	125 kHz
spectrum	438.050–438.175 MHz	439.550–439.675 MHz	434.425–434.550 MHz
spreading factor SF	11	11	11
code rate CR	1 (5/4)	1 (5/4)	1 (5/4)
preamble sync length	8 symbols	8 symbols	8 symbols
preamble sync word	`0x12`	`0x12`	`0x12`
header mode	explicit (20 bits)	explicit (20 bits)	explicit (20 bits)
CRC	on (16 bits)	on (16 bits)	on (16 bits)
IQ polarisation	normal	inversed	inversed

Note: Above preferred frequencies are outside the interfering 433—435 MHz ISM band and mostly respect the IARU Region 1 70 cm band plan.

Attention: The alternative downlink frequency is only intended for those countries where amateur radio is not allowed to transmit (e.g. Austria) or has secondary status (e.g. The Netherlands) on the preferred downlink frequency. **Terminal-type end devices will expose an option to select the alternative downlink.**

I-gates using the alternative downlink frequency are fervently advised to emit at higher power levels (e.g. 7.5 W) to overcome the interference caused by ISM systems.

Note:

- In order to achieve a maximum range, Semtech — the company that developed LoRa — recommends selecting the maximum spreading factor $SF = 12$. However, SF12 is extremely slow, offering only a mere 36.6 byte/s.
- Likewise, the bandwidth is set to the smallest commonly available bandwidth among all LoRa ICs, namely $BW = 125\,\text{kHz}$. This is by design also the chip rate $R_c = BW$.
- To avoid any further overhead to an already slow mode, the forward error correction (FEC) code rate is kept at $CR = 1$, which corresponds to $\frac{data}{data+FEC} = \frac{4}{5}$.
- It was observed that amateur radio predominantly employs the LoRa sync word `0x12`; which is manufacturer recommended for private networks, and differs from the `0x34` for a LoRaWAN.

With spread-spectrum modulation, a symbol (or chirp with LoRa) consists out of many chips. The spreading factor SF is defined as the number of raw bits per symbol. Hence, each symbol or chirp holds $2^{SF} = 2^{11} = 2048$ chips.

This allows one to calculate the symbol rate R_s from the chip rate R_c:

$$R_s = \frac{R_c}{2^{SF}} = \frac{BW}{2^{SF}} = \frac{125\,000}{2^{11}} \approx 61\,\text{symbols/s}$$

The effective data rate DR or bit rate R_b can be obtained by taking into account the forward error correction:

$$DR = R_b = R_s \cdot SF \cdot \frac{4}{4 + CR} = \frac{125\,000}{2^{11}} \cdot 11 \cdot \frac{4}{5} \approx 537\,\text{bits/s} \approx 67\,\text{byte/s} \tag{1.1}$$

Above parameters are adequate for sending LoRa frames with short, compressed payloads over the almost longest possible distance when the number of participant nodes is relatively low.

See also:

For an in depth tutorial slide series about LoRa (and LoRaWAN), please refer to Mobilefish.com, also available in video format on YouTube.

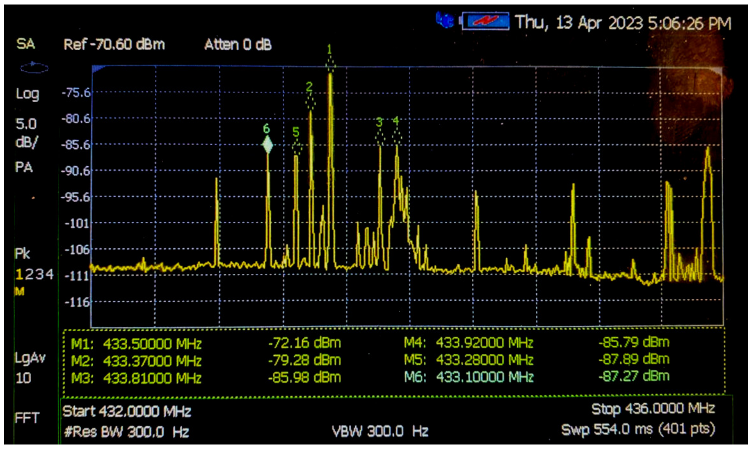
1.1 Why 438 MHz

1.1.1 Uplink

In many ITU Region 1 countries, the ISM-band extends from 433.050 to 434.790 MHz. Amateur radio services of these countries operating within this band must accept harmful interference which may be caused by these applications, as noted in the ITU amateur footnotes. A similar situation exists in the other ITU regions, see Figure 1.1.

To protect weak uplink signals from this ISM interference, **438.050–438.175 MHz** was selected as the APRS 438 uplink spectrum. This use is compliant with the IARU Region 1 UHF band plan.

Distance is also kept from Austrian POCSAG on 438.025 MHz and most of the Austrian 70 cm repeaters, with the exception of a handful of AFSK digipeaters.



```
M1: 433.50000 MHz   -72.16 dBm    M4: 433.92000 MHz   -85.79 dBm
M2: 433.37000 MHz   -79.28 dBm    M5: 433.28000 MHz   -87.89 dBm
M3: 433.81000 MHz   -85.98 dBm    M6: 433.10000 MHz   -87.27 dBm
Start 432.0000 MHz                                 Stop 436.0000 MHz
#Res BW 300.0 Hz          VBW 300.0 Hz         Swp 554.0 ms (401 pts)
```

Figure 1.1: The ISM-band spectrum in Santiago de Chile, a metropole with seven million inhabitants.

1.1.2 Downlink

The default downlink spectrum is **439.550–439.675 MHz.** This is compliant with the IARU Region 1 UHF band plan.

1.1.3 Alternative Downlink

The default downlink spectrum is problematic in at least two countries:

- In The Netherlands, amateur radio has no primary status in the band segment 436–440 MHz.
- In Austria, the band segment starting from 439.100 MHz is receive only.

To deal with these limitations, an alternative downlink spectrum is suggested: **434.425–434.550 MHz.**

This is bandwidth-wise not compliant with the IARU Region 1 UHF band plan, but is still within a band segment reserved for digital communications. However, in these countries the IARU Region 1 band plan is not upheld anyhow.

Attention: The alternative downlink spectrum is only intended for those countries where amateur radio is not allowed to transmit (e.g. Austria) or has secondary status (e.g. The Netherlands) on the preferred downlink spectrum. **Terminal-type end devices will expose an option to select the alternative downlink.**

I-gates using the alternative downlink spectrum are fervently advised to emit at higher power levels (e.g. 7.5 W) to overcome the interference caused by ISM systems.

1.1.4 ITU Regulation

From an International Telecommunication Union (ITU) regulatory point of view, long range communication —which, by definition, includes LoRa (= “Long Range”)— is not allowed on ISM (Industrial, Scientific & Medical) bands. ISM bands are intended for local use only.

The amateur radio service forms a sole exception to this, as its 70 cm UHF band happens to overlap the ITU Region 1 433–435 MHz ISM band. In most countries, amateur radio as a primary service over most of the 70 cm band. Moreover, ham radio is not restricted to a 20 dBm (= 100 mW) power level, nor any 1% duty cycle limits on this band.

Tip: **The modulation gain of LoRa over FSK is about 10 dB** in the link budget. By consequence, a 10 W AFSK packet link could be replaced by a 1 W LoRa link.

Caution: As a general rule, secondary users should always check whether a frequency is in use by a primary user before transmitting on air. However, **LoRa has no carrier sensing capability.** Therefore, secondary (ISM) band users lack the ability to check whether a primary service is using the 70 cm band.

1.2 Why SF11

Depending on how popular APRS over LoRa becomes and on how intensely it will get used, there might come a time when the LoRa channel gets saturated. Unlike packet radio, LoRa has no carrier sensing capability. Sending longer text messages, even when compressed, may aggravate the situation.

In order to prevent such a congestion, APRS 438 decided to **only employ SF11.**

Doing so, effectively doubles the data rate over SF12. It also saves 50% on airtime and batteries. The slight range penalty from switching from SF12 to SF11 is in most circumstances absolutely acceptable, provided the density of i-gates in an area is sufficient.

With a payload of only 17 bytes, the compressed geolocation frame is perfectly geared towards taking advantage of the reduced airtime offered by SF11 (see Figure 2.1).

Unfortunately, most cheap i-gates currently in use by ham operators are only capable of receiving one preset spreading factor. Therefore, the choice was made to use SF11 exclusively. Considering what some members of the amateur radio community have come to expect of LoRa, the faster data rate offered by SF11 is more than warranted.

CHAPTER

TWO

APRS FRAME COMPRESSION

LoRa, as a physical layer, permits sending any of the 256 characters from `\00` to `\ff`; double the amount of the 7-bit, 128 ASCII character set. AX.25 (packet radio) unnumbered information (UI) frames at the data link layer are no different in this respect.

However, as *previously demonstrated*, the effective data rate of LoRa SF11 is much slower than what can be achieved with 1200 baud packet. Hence, the need to compress data with LoRa is more urgent.

AX.25 UI frame field	compression opportunities
Flag	**not required**; explicit header provided by LoRa
Destination Address	**not required**; software version provided by the i-gate
Source Address	6 out of **37** possible characters: 26 capital letters + 10 digits + space
SSID	1 out of **16** hexadecimal digits
Digipeater Address	any out of ***5*** *recommended* `n-N` *paradigm paths*
Control Field	**not required**
Protocol ID	**not required**
Information Field	256 characters of which **95** printable ASCII charactersfirst character = Data Type ID
Frame Check Sequence	**not required**; FEC & CRC are provided by LoRa
Flag	**not required**

Note:

- *Source Address, SSID* and *Data Type ID* can be compressed into only 5 payload bytes, compared to 26 payload bytes with OE5BPA firmware.
- It is customary to compress latitude, longitude, symbol, course and speed using Base91, which results in another 13 payload bytes; *Data Type ID* not included. **APRS 438** will not differ in this respect.
- If APRS Mic-E compression were to be used instead, that would require another 16 payload bytes to compress latitude, longitude, symbol, course and speed; 7 bytes in the superfluous *Destination Address* and 9 bytes in the *Information Field; Data Type ID* included. Hence, this is not a good option.

2.1 Measurable Benefits

2.1.1 Reduced Packet Error Rate

APRS 438 geolocation beacons will encode a total of **only 17 payload bytes** at a time, tremendously **increasing the chances of a flawless reception** by an APRS 438 LoRa i-gate. Other firmware tends to consume about six times as many LoRa payload bytes.

Important: LoRa may receive up to 20 dB under the noise floor. However, keep in mind that **the packet error rate (PER)** as a function of the bit error rate (BER) **increases with the number of transmitted bits**.

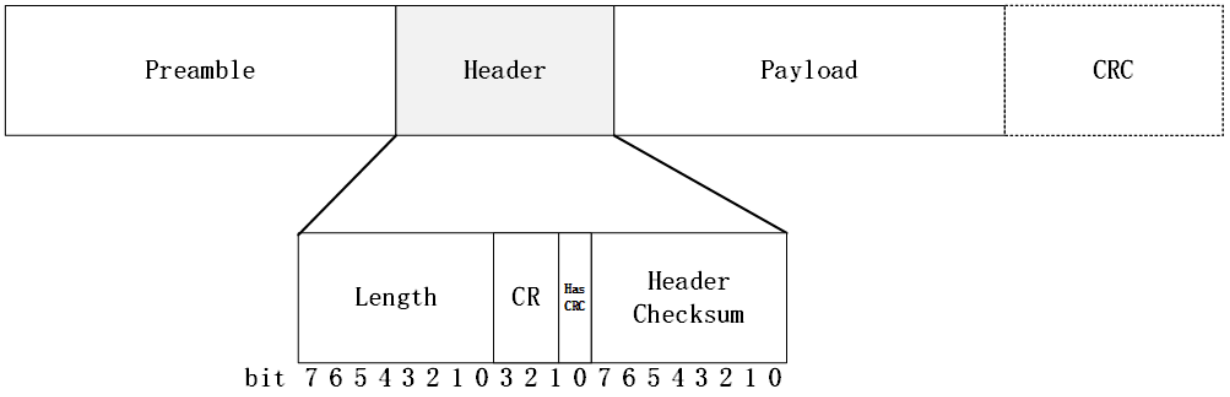
$$PER = 1 - (1 - BER)^n \approx n \cdot BER$$

approximately, when BER is small and n is large, and where:

- $(1 - BER)$: the probability of receiving a bit correctly
- n: the number of bits in a packet; which is 8 times the number of bytes

PER Examples

When used with an explicit header (recommended), **LoRa packets will have the following 36 bit overhead:** a 20 bit physical header and a 16 bit payload cyclic redundancy check (CRC) at the end of the packet.



With that knowledge, take for example a bit error rate BER of one in thousand. This would result in the following packet error rates:

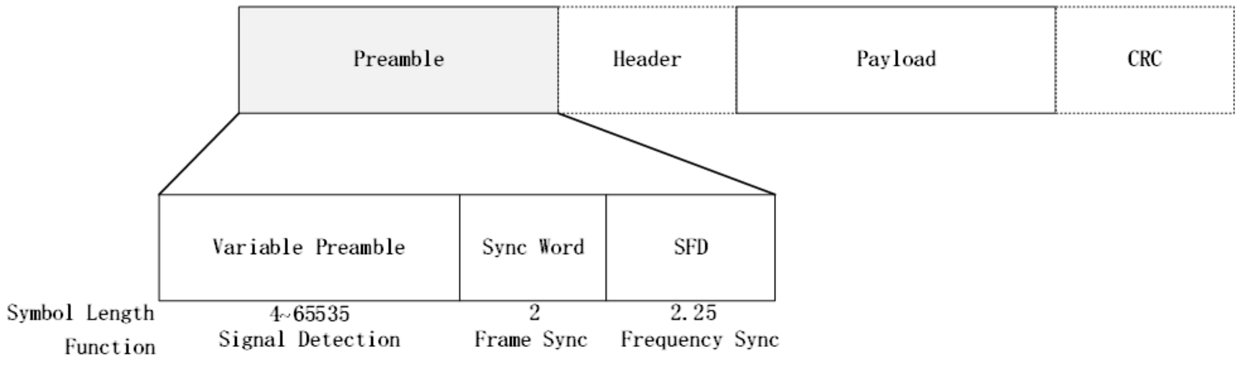
payload	17 bytes	24 bytes	28 bytes	45 bytes	113 bytes
overhead	36 bits	36 bits	36 bits	36 bits	36 bits
n	172 bits	228 bits	260 bits	396 bits	940 bits
BER	0.1%	0.1%	0.1%	0.1%	0.1%
PER	15.8%	20.4%	22.9%	32.7%	61.0%

By consequence, the chances of correctly receiving a 17 byte payload are more than twice as high as with a 113 byte payload:

$$\frac{1 - 0.158}{1 - 0.610} \approx 2.18$$

Note: In reality, above calculations are more convoluted as LoRa employs symbols that are chip jumps or discontinuities in chirps to convey information.

Moreover, the explicit header is preceded by a preamble. The preamble consists out of a configurable length preamble, a set sync word, a start frame delimiter (SDF) and a small pause preceding the explicit header.



The variable preamble is important as it trains the receiver at receiving the signal. Hence, the symbol length of this variable preamble also has an effect on the packet error rate.

See also:

Details about the LoRa packet structure can be found here.

2.1.2 Airtime Reduction

Keeping the payload as small as possible, has an even more important reason: to reduce the airtime required to send the LoRa frame. As calculated in equation (1.1), **LoRa SF11 is a slow data rate mode.** Reducing the airtime also **saves battery power** on portable devices.

Due to the LoRa symbol encoding scheme, airtime reductions occur **in abrupt steps of 4 payload bytes** when the spreading factor is SF11 and the bandwidth 125 kHz (CR=1, explicit header, CRC=on). This is depicted by the second stepped trace from the top in Figure 2.1 (adapted from airtime-calculator).

Table 2.1: Airtime of common payloads as a function of spreading factor SF

payload	5 bytes	17 bytes	24 bytes	28 bytes	45 bytes	113 bytes
airtime with SF12	0.83 s	1.32 s	1.48 s	1.65 s	2.14 s	4.43 s
airtime with SF11	0.50 s	0.66 s	0.82 s	0.91 s	1.15 s	2.46 s
airtime with SF10	0.25 s	0.33 s	0.37 s	0.41 s	0.56 s	1.23 s

See also:

The Things Network (TTN) organisation, albeit a global LoRaWAN, is exemplary in stressing the importance of maintaining LoRa payloads small.

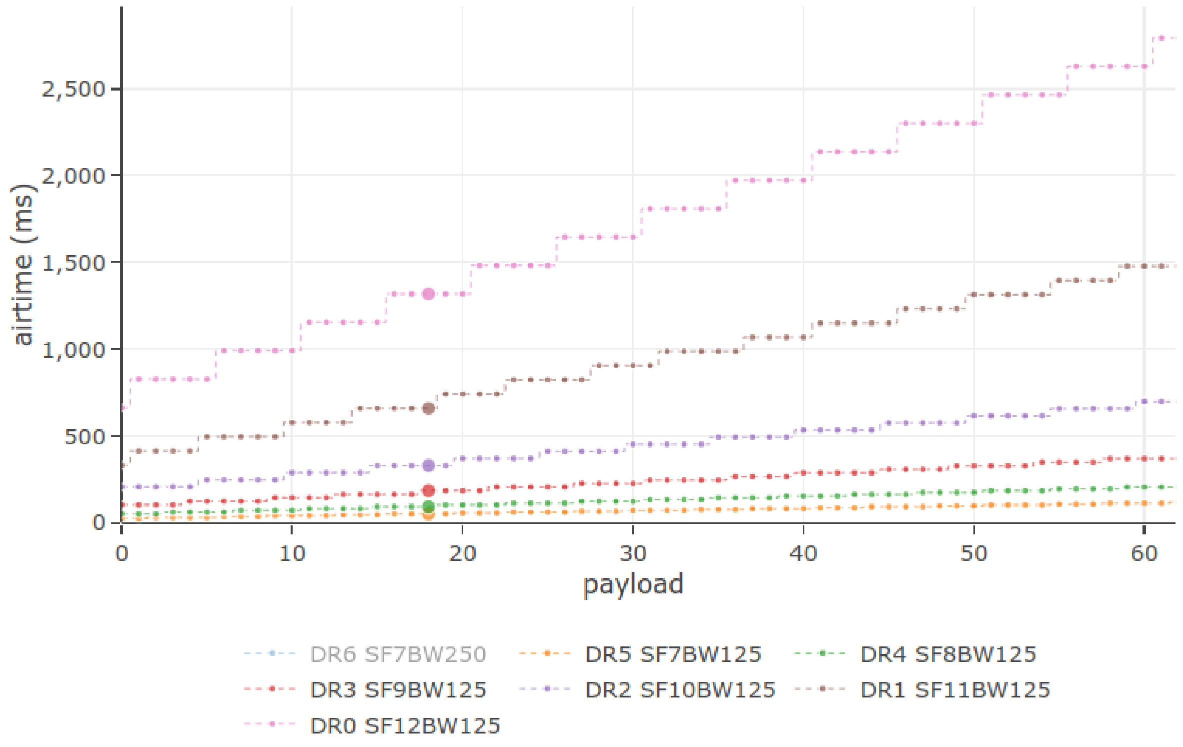


Figure 2.1: The top trace is for SF12BW125. The dot represents a total payload of 17 bytes as proposed for geolocation packets with compression.

2.2 Callsign, SSID, Path and Data Type Compression

Callsigns contain only capital letters, digits and empty spaces. Up to six characters from such a 37 character set can easily be compressed into 4 `CCCC` bytes of the extended 256 character set.

Hence, all compressed APRS frames in this standard begin with 5 `CCCCD` bytes, irrespectively of the Data Type. Furthermore, **the compressed frame length is voluntarily limited by design to maximum 45 bytes,** which leaves up to 40 bytes of payload. For certain *Data Types,* the maximum length is even significantly lower. A maximum frame length of 45 bytes corresponds to a maximum airtime of 1.15 s with SF11.

Important: I-gates should test whether the payload length of a received frame is in correspondence to the declared *Data Type*. **Frames that do not comply, should be rejected.**

The combination of the four first `CCCC` Base256 *Callsign* bytes and the in `D` declared *Data Type* with the corresponding payload length form **the key** —so to speak— to the i-gate. This is what allows for a headerless frame design. It prevents the i-gate from relaying frames that are not intended for this compressed frame link.

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	40 bytes
`CCCC`	`D`	

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code (between type 0 and 3; included)

2.2.1 Base37

Base37 consist out of the following 37 ordered digits, starting with a space character:

```
digits = ' 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

2.2.2 Encoding CCCC

1. Perform input sanitisation and right padding with spaces up to 6 characters.
2. Treat the given 6 character callsign string as a Base37 encoding. Decode it first to an integer.
3. Then, encode this integer as a 4 byte Base256 `CCCC` bytestring.

2.2.3 Decoding CCCC

1. First, decode the given 4 byte Base256 `CCCC` bytestring to an integer.
2. Then, encode this integer as a 6 character Base37 string, corresponding to the callsign.

2.2.4 Encoding D

1. Perform input sanitisation.
2. Multiply the *SSID* integer by 16.
3. Multiply the *Path Code* by 4.
4. Then, algebraically add to these intermediate results to the *Data Type Code* integer from Table 2.3.
5. Finally, convert the resulting integer to a single Base256 `D` byte.

2.2.5 Decoding D

1. First, decode the given Base256 `D` byte to an integer.
2. The *SSID* equals the integer quotient after integer division of the decoded integer by 16.
3. The remainder of above integer division is subjected to a second integer division by 4.
4. The *Path Code* equals the integer quotient of this second integer division.
5. Whereas the *Data Type Code* equals the remainder this second integer division.

2.2.6 SSID Recommendations

A secondary station identifier is a number in the range 0-15, as an adjunct to the station *Callsign*. Similarly as with IEEE 802.11 wireless networks, an APRS SSID identifies a set of APRS station capabilities.

Table 2.2: SSID recommendations

SSID	APRS station type
0	primary station; usually fixed and message capable
1	generic additional station, digi, mobile, wx, etc.
2	generic additional station, digi, mobile, wx, etc.
3	generic additional station, digi, mobile, wx, etc.
4	generic additional station, digi, mobile, wx, etc.
5	other networks (D-STAR, DMR, smartphones etc.)
6	special activity, satellite ops, camping, etc.
7	walkie talkies, HTs or other human portable
8	boats, sailboats, RVs or second main mobile
9	primary mobile (usually message capable)
10	Internet, **(LoRa) i-gates,** echolink, Winlink, AVRS, APRN, etc.
11	balloons, aircraft, spacecraft, etc.
12	APRStt, DTMF, RFID, devices, **one-way (LoRa) trackers,** etc.
13	weather stations
14	truckers or generally full time drivers
15	generic additional station, digi, mobile, wx, etc.

Tip: One-way trackers best use the 12 one-way *SSID* indicator, whereas *SSID* 9 usually means a ham with full communication capabilities; both APRS message and voice.

2.2.7 Data Type Codes

Of all the *Data Types* defined in the APRS Protocol Reference, a subset was selected, based on popularity and suitability for LoRa.

Table 2.3: Data Type Codes

Data Type	*ID* (not used)	*Data Type Code*	payload
compressed ***geolocation*** — no timestamp	! or =	0	17 or 19 bytes
complete ***weather report*** — with compressed geolocation, no timestamp	! or =	0	28 or 29 bytes
status report (28 characters)	>	1	6—24 bytes
item report — with compressed geolocation	)	2	20—24 bytes
addressed message (51 characters)	:	3	10—45 bytes

Note:

- APRS 438 will not transmit any *ID* byte over LoRa. The *ID* will be added at the i-gate.
- Weather reports use the same *IDs* and *Data Type Codes* as position reports but with a *Symbol Code* _ overlay.
- A *Symbol Table Identifier* nor a *Symbol Code* can be compressed.

2.2.8 Path Codes

The path codes are of little importance to LoRa APRS 438. Path codes mainly serve to instruct (VHF) APRS digipeaters. These digipeaters may be co-located with a LoRa i-gate or may obtain packets from Internet APRS-IS. See also *No Digipeating on the Uplink*.

Table 2.4: Data Type Codes

station	recommended `n-N` paradigm path	*Path Code*
no digipeating		0
metropolitan fixed, mountain expeditions, balloons & aircraft	`WIDE2-1`	1
extremely remote fixed	`WIDE2-2`	N/A
metropolitan mobile	`WIDE1-1,WIDE2-1`	2
extremely remote mobile	`WIDE1-1,WIDE2-2`	N/A
space satellites	`ARISS,WIDE2-1`	3

Important:

- The first `n` digit in `n-N` paradigm paths indicates the coverage level of the digipeater, whereby `1` is for local fill-in digipeaters and `2` is for county-level digipeaters.
- The second `N` digit indicates the number of repeats at the indicated coverage level.

2.3 Compressed Geolocation Frames

A compressed geolocation frame has a payload of either exactly **17 or 19 bytes.**

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	12 (or 14) bytes
`CCCC`	`D`	`/XXXXYYYY$cs(aa)`

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code = 0
- `/`: the *Symbol Table Identifier*
- `XXXX`: the Base91 compressed longitude
- `YYYY`: the Base91 compressed latitude
- `$`: the *Symbol Code*
- `cs`: the compressed course (in degrees) and speed (in knots)
- `(aa)`: optionally, the compressed altitude (in feet)

Note:

- Terrestrial objects do not require sending altitude data. Anyhow, GPS height readings are notorious for being significantly inaccurate.
- APRS-IS already understands Base91 `XXXXYYYY` compression when altitude `aa` is not used.
- In absence of altitude `aa`, the i-gate adds the *Compression Type Byte* `T` right behind `cs`.
- When `aa` is present, the i-gate will instead need to decompress the whole frame and forward the uncompressed frame to APRS-IS.
- The parenthesis are not sent; these merely indicate optionality.

2.4 Compressed Weather Report Frames

A compressed weather report frame has a payload of either exactly **28 or 29 bytes.**

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	23 (or 24) bytes
`CCCC`	`D`	`/XXXXYYYY_csgtrrppPPhbb(S)`

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code = 0
- `/`: the *Symbol Table Identifier*
- `XXXX`: the Base91 compressed longitude
- `YYYY`: the Base91 compressed latitude
- `_`: the weather report *Symbol Code*
- `cs`: the compressed wind direction (in degrees) and sustained one-minute wind speed (in knots)
- `g`: gust (half of peak wind speed in km/h in the last 5 minutes)
- `t`: temperature (in kelvin above 173.15 K)
- `rr`: rainfall (in mm) over the past hour
- `pp`: rainfall (in mm) over the past 24 hours
- `PP`: rainfall (in mm) since midnight
- `h`: humidity (in %)
- `bb`: barometric pressure (in Pa above 50000)
- `(S)`: optionally, snowfall (in cm) over the past 24 hours

Notes:

- All numerical encodings are one or two byte Base256 encodings.
- Here is a fascinating list of weather records.
- The i-gate adds the *Compression Type Byte* `T` right behind `cs`.
- The parenthesis are not sent; these merely indicate optionality.

2.5 Compressed Text

In order to prevent channel congestion, it is crucial to limit the character set of messages. This allows for more frame compression. In resemblance to Morse code, the character set would contain only 26 Latin capital letters, the 10 digits, space and a few punctuation marks and symbols. Limiting the set to 42 characters makes it fit 6 times in the 256 character set of LoRa.

character subset	# of characters
Latin capital letters	26
digits	10
space	1
punctuation `.` `?`	2
symbols `-` `/` `@`	3
TOTAL	**42**

2.5.1 Base42

Base42 consist out of the following 42 ordered digits, starting with a space character:

```
digits = ' 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ-./?@'
```

2.5.2 Encoding tttt

1. Perform input sanitisation.
2. Perform character replacement and filtering on the given string; only allow for characters of the *42 character set*.
3. Treat the resulting text string as a Base42 encoding. Decode it first to an integer.
4. Then, encode this integer as a Base256 `tttt` bytestring.

2.5.3 Decoding tttt

1. First, decode the given Base256 `tttt` bytestring to an integer.
2. Then, encode this integer as a Base42 string, corresponding to the text.

Caution: REFRAIN from adding any APRS comments! Adding more bytes to a LoRa frame only reduces the chances on successful reception. Rather, consider sending an occasional *status report*.

2.6 Compressed Status Report Frames

A compressed status report frame has a payload of between **6 and 24 bytes.**

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	1—19 bytes
`CCCC`	`D`	`t(tttt...tttt)`

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code = 1
- `t(tttt...tttt)`: between 1 and 19 bytes of compressed text from a limited 42 character set, corresponding to a maximum of **28 uncompressed characters**

2.7 Compressed Item Report Frames

A compressed item report frame has a payload of between **20 and 24 bytes.**

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	15—19 bytes
`CCCC`	`D`	`/XXXXYYYY$csTttt(tttt)`

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code = 2
- `/`: the *Symbol Table Identifier*
- `XXXX`: the Base91 compressed longitude
- `YYYY`: the Base91 compressed latitude
- `$`: the *Symbol Code*
- `cs`: the compressed course and speed
- `ttt(tttt)`: 3 to 7 bytes for the compressed *Item Name* (**between 3 and 9 characters** of the limited 42 character set)

Notes:

- The i-gate adds the *Compression Type Byte* `T` right behind `cs`.
- The parenthesis are not sent; these merely indicate optionality.

2.8 Compressed Addressed Message Frames

Up to now, APRS has been unduly considered to be predominantly a one-way localisation technology. This went to the point that many would mistake the letter "P" in the APRS acronym for "position" instead of "packet." Bob Bruninga WB4APR (SK), the spiritual father of APRS, deeply resented this situation.

> *"APRS is not a vehicle tracking system. It is a two-way tactical real-time digital communications system between all assets in a network sharing information about everything going on in the local area."*

In Bob's view of APRS as being foremost a real-time situational and tactical tool, addressed messaging definitely merits its place. Our goals is to render APRS messaging more popular by offering LoRa messaging pager terminals.

A compressed addressed message frame has a payload of **between 10 (for an empty ping) and 45 bytes.** The available message length of 51 characters is largely sufficient for, for example, SOTA self-spotting using APRS2SOTA.

Callsign	*SSID,Path Code &Data Type Code*	*Compressed Data*
4 bytes	1 byte	5—40 bytes
`CCCC`	`D`	`EEEEF(tttt...tttt)`

where:

- `CCCC`: 4 bytes for the compressed **6 character** *Callsign*
- `D`: compresses into 1 byte:
  - the SSID (between SSID 0 [none] and 15; included),
  - the Path Code (between path 0 [none] and 3; included), and
  - the Data Type Code = 3
- `EEEE`: 4 bytes for the compressed *Addressee* (up to 6 character callsign)
- `F`: compresses into 1 byte:
  - the *Addressee SSID* (between SSID 0 [none] and 15; included), and
  - the *Message No* (from 0 to 15; included)
- `(tttt...tttt)`: up to 35 bytes of compressed text from a limited 42 character set, corresponding to a maximum of **51 uncompressed characters**

2.8.1 Encoding and Decoding EEEE

The `EEEE` codec algorithms are identical to the *CCCC codec algorithms*.

2.8.2 Encoding F

1. Perform input sanitisation, i.e. perform a modulus 16 operation on a *Message No* originating from APRS-IS.
2. Multiply the *SSID* integer by 16.
3. Then, algebraically add to this the *Message No* integer.
4. Finally, convert the resulting integer to a single Base256 `F` byte.

2.8.3 Decoding F

1. First, decode the given Base256 `F` byte to an integer.
2. The *SSID* equals the integer quotient after integer division of the decoded integer by 16.
3. Whereas the *Message No* equals the remainder of the decoded integer by 16 (modulo operation).
4. When relaying the text message to APRS-IS, the i-gate will add the last digit of the minute the text message was received in front of the received *Message No*.

2.9 Codec Algorithms

Note: This codec is **not** an encryption algorithm, since it is openly published here. Therefore, this codec is perfectly legal for use under amateur radio regulations.

2.9.1 Python3 Implementation

Important: This is the **reference implementation.** Other implementations should yield identical results as the test examples contained in the Python3 implementation.

- Python3 codec algorithms and tests

2.9.2 C Implementations

- C for PC codec algorithms and tests
- Arduino C for ESP32 codec algorithms (under development)

CHAPTER

THREE

I-GATE

3.1 I-Gate Functionality

I-gates will, in the listed order of priority, and with respect for the received *Path Code:*

1. **Reject** any received frame which payload length does not correspond to the declared *Data Type*.
2. Keep a **list of all stations heard** on the uplink channel over the last hour. This implies that all end devices need to send a geolocation frame or at least a status report when switched on and when not having transmitted over the course of an hour.
3. Transmit on the downlink channel all frames heard on the uplink channel, **other than addressed text messages.**
4. Transmit **text messages addressed to end devices the i-gate heard recently.** These addressed text messages may originate from the uplink, from an attached packet APRS digipeater or from Internet APRS-IS.
5. Transmit **situational awareness frames** from an attached packet APRS digipeater or APRS-IS, when these are applicable to the geographical coverage area of the i-gate.

When the downlink channel gets saturated, above listed order of priority applies in terms of dropping frames or keeping a frame wait list stack.

3.2 No Digipeating on the Uplink

Caution: REFRAIN from digipeating on the uplink frequency!

Since LoRa SF11 is a slow data rate mode, digipeating on the LoRa uplink channel quickly leads to unwanted channel congestion. Unlike AX.25 packet radio, LoRa does not offer carrier sensing (CS); only channel activity detection (CAD).

Also consider that:

- LoRa was merely intended as an **Internet access technology.**
- Most LoRa gateways are connected to the APRS-IS Internet server network and many users are merely interested in reaching APRS-IS.
- There are hardly any low power portable LoRa devices able to display situational awareness in relation to other LoRa devices.
- Only in extremely remote areas without Internet access, digipeating may be considered, but only on the downlink frequency or the alternative downlink frequency.

Important: Hence, `n-N` paradigm paths could be interpreted foremost as crossover AX.25 packet digipeating paths for any (VHF) digipeater co-located with the LoRa (i-)gate.

Hint: However, suppose meshing or `n-N` paradigm digipeating were to be allowed on a single LoRa channel; even for trackers. This would offer interesting emergency capabilities when no Internet is available. However, this would absolutely require switching from SF11 to the higher data rate offered by SF10, as demonstrated in Table 2.1. In such a scenario, *Path Code* stands for the LoRa device communicating its digipeating requirements to the mesh network.

CHAPTER

FOUR

PROTOCOL REVISIONS

date	changes
2022-05	initial release
2023-03	change in the order of the codec digits; also, `@` instead of `_` is now allowed in text messages
2023-07	separate up- and downlinks, SF11, i-gate functions

CHAPTER

FIVE

ROAD MAP

5.1 Tracker

firmware	completed	feature	payload	compatible with OE5BPA i-gate
v0.0.0	✓	original OE5BPA tracker	113 bytes	✓
v0.1.0	✓	byte-saving `tracker.json`	87 bytes	✓
v0.2.0	✓	fork of the OE5BPA trackerwith significantly less transmitted bytes	44 bytes	✓
v0.3.0	✓	Base91 compression of location, course and speed data	31 bytes	✓
`v0.4.0`	✓	removal of the transmitted newline `\n` character at frame end	30 bytes	✓
v1.0.0	Q3 2023	frame compression	17 bytes	Use the APRS 438 i-gate!

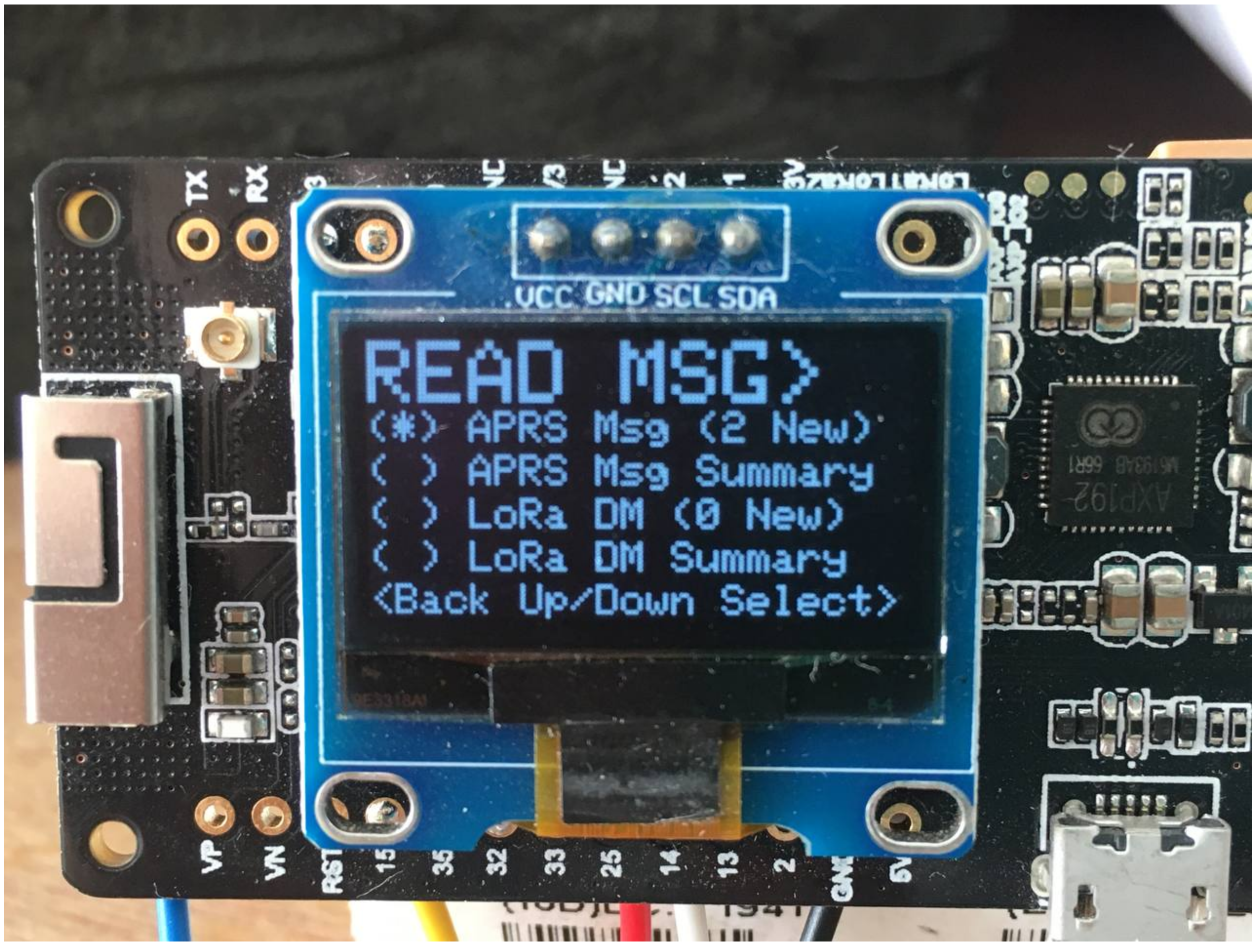
Note: Currently, the APRS 434 tracker is still compatible with the i-gate developed by Peter Buchegger, OE5BPA.

Important: The release of frame compression will require migration to *new frequencies, SF11* and the use of the APRS 438 i-gate.

Compressed geolocation frames are already flawlessly received by the APRS&438 terminal and being forwarded to APRS-IS. The release of tracker firmware is kept on hold until the i-gate firmware gets released.

5.2 Terminal

Currently under development.



Ricardo Guzmán Christie, CD2RXU, brought this project in a well advanced stage. Together with Matthias Brändli, HB9EGM, both are now working on the ESP32 Arduino C++ implementation of the text compression algorithm.

5.3 I-Gate

Currently under development.

Works, but needs some more attention and care. Manfred Heindl, DC2MH, recently offered Ricardo help with this.

CHAPTER

SIX

FUTURE PLANS

6.1 From Pure to Reservation ALOHA

Recent academic research investigated the collision of two LoRa packets of equal power when collided with a start time offset. It was found that first LoRa packet stands a high chance of being correctly received as long as its cyclic redundancy check (CRC) information was not interfered by the second LoRa packet. This very sensitive CRC information of a LoRa packet is sent in the explicit header, towards the beginning of the packet.

Most LoRa APRS implementations allow end devices to transmit at any moment of time. This inevitably results in packet collisions and the loss of the information of one or more packets. This situation is called pure ALOHA (P-ALOHA). It is comparable to a pile up during a DXpedition.

Above situation can be improved upon by allowing the packet transmissions to start only at well defined moments in time. This form of medium access control (MAC) is called slotted ALOHA (S-ALOHA). However, even with slotted ALOHA, packet collisions can still occur.

Taking this a step further, with reservation ALOHA (R-ALOHA) the i-gate temporarily assigns a time slot to an end device that was successful in reaching the i-gate in one of the vacant time slots. With this MAC scheme, packet collisions now can only occur at first access; resulting in a vastly more efficient use of the channel. This scheme is comparable to a controlled HF net.

pure ALOHA	slotted ALOHA

CHAPTER

SEVEN

FIRMWARE

Tip: If you prefer to write your own firmware, please, check out the *codec algorithms*.

7.1 Tracker Firmware

Currently, we are offering `tracker firmware v.0.4.0` that is still compatible with the i-gate developed by Peter Buchegger, OE5BPA.

This tracker **only sends 30 bytes of payload data,** instead of the usual 113 bytes.

See *Road Map* for further developments.

7.2 Terminal Firmware

Currently under development; see *Road Map*.

7.3 I-Gate Firmware

Currently under development; see *Road Map*.

CHAPTER

EIGHT

HARDWARE

8.1 LoRa ICs and Modules

- Semtech LoRa products
- Semtech SX1278
- HopeRF LoRa modules
- HopeRF RFM98W
- G-NiceRF Lora1278F30 1 watt module

8.2 Tracker Hardware

- TTGO T-Beam 433 MHz v0.7 or v1.1
- longer 433 MHz antenna with SMA male connector
- 16.9 mm long tiger tail wire soldered to the female SMA socket
- 5 V, 3 A USB charge adapter with appropriate microUSB or USB-C cable
- Panasonic NCR18650B Li-ion cell, or quality equivalent
- glue gun to stick the GPS antenna to the cell holder
- SH1106 1.3" I^2C (4-pin) OLED display (slightly larger than the usual 0.8" displays often sold with the TTGO T-Beam)
- enclosure

8.3 I-Gate Hardware

- Either:
  - TTGO LORA32 433 MHz v2 (U.FL or SMA female RF socket), or
  - maybe Heltec ESP32 LoRa 433 MHz **v2** (U.FL female RF socket); subject to satisfactory receiver testing
- 70 cm amateur radio colinear groundplane antenna with coaxial cable and connectors
- 16.9 mm long tiger tail wire soldered to the RF socket
- 5 V, 1 A USB power supply with appropriate microUSB or USB-C cable

- enclosure

Caution: **DO NOT USE** Heltec ESP32 LoRa 433 MHz **v1** as it is as deaf as a post!

CHAPTER

NINE

NEWS, SOCIAL & CO-DEVELOPMENT

Feel free to join our public **Telegram Group** for the latest news and cordial discussions.

You are invited to contribute code improvements to **this project on GitHub**. Here is a lightweight video introduction to using GitHub by Andreas Spiess, HB9BLA.

CHAPTER

TEN

ACKNOWLEDGEMENTS

10.1 Firmware

- Ricardo Guzmán Christie, CD2RXU, for developing terminal and i-gate firmware employing the compression algorithms presented in this white paper.
- Bernd Gasser, OE1ACM, for the earliest LoRa APRS experiments and code
- Christian Johann Bauer, OE3CJB, for the Base91 geolocation compression algorithm
- Peter Buchegger, OE5BPA, for providing a tracker and i-gate firmware as open source code, in a handy PlatformIO environment, with over-the-air (OTA) i-gate updates. This was the ideal starting point for running LoRa frame compression experiments.

10.2 Codec

- Serge Y. Stroobandt, ON4AA, for devising the protocol and Python codec algorithms, as well as initiating this project by writing the protocol white paper.
- Folkert Tijdens, PA0FOT, for contributing *`codec.cpp`* and asking the right questions, rendering this document more scholarly
- Matthias Brändli, HB9EGM, for contributing the Arduino C implementation of the `tttt` codec algorithm.
- Pascal Schiks, PA3FKM, for providing insights about microcontroller stacks

10.3 Spectrum Selection

- Wolfgang Hallmann, DF7PN, for informing that, in a number of European countries, the ISM-band extends from 433.05 to 434.79 MHz.
- Gerhard Hickl, OE3GHB, for pointing out that, in Austria, the spectrum above 439.1 MHz is receive only.

10.4 Testing

- Erwin Fiten, ON8AR, for testing firmware and reporting on long distance car approaches to the LoRa i-gate
- Jan Engelen, DL6ZG, for testing firmware and providing feedback
- Greg Stroobandt, ON3GR, for cycling around the city with a privacy invading tracker

10.5 Infrastructure

- ReadTheDocs.org for hosting the documentation of this project, free of charge
- Github.com for hosting the project source files, free of charge
- The Sphinx documentation generator and its extensions
- executable{books} for the Markedly Structured Text MyST Python parser (cheat sheet, syntax extensions)