ONE NET

wireless control for everyone

ONE-NET SPECIFICATION

Version 2.3.0

August 14, 2012

Table of Contents

1 Overview

ONE-NET provides the open design specification necessary to enable a low-cost, low-bandwidth wireless control network. Many residential applications need control/automation devices to be low cost, very low power (able to run on batteries), and able to operate in secure wireless communication modes. For the greatest market penetration, these devices must be interoperable, using open design standards.

1.1 Scope

ONE-NET defines a standard physical, network, and message protocol primarily, though not exclusively, designed to meet the needs and capabilities of low-cost microcontrollers and transceivers with minimal resources (typically, sensors and controllers used in a home or small office environment). In particular, ONE-NET is aimed towards networks where the majority of devices are very simple, low-cost battery-operated client devices and the primary concern is low power consumption. However, ONE-NET also provides support for networks containing more advanced devices and / or devices where low power consumption is not the primary concern.

ONE-NET is a user-supported technology.

2 Nomenclature

The following sections define terms used in this document.

2.1 Definitions

Block Transaction: A transfer of a fixed number of data bytes that do not fit into a single data packet, and are delivered reliably.

Channel: A given frequency that ONE-NET Devices use to communicate with one another.

Channel Back-Off Time: The time (5ms) that a Device waits, after sensing that the channel is busy, before it tries sensing the channel again.

Client: Any Device that is not a ONE-NET Master.

Client Repeater: A Client that listens for multi-hop packets and retransmits them. Such Devices are generally powered on at all times.

Repeater: Same as Client Repeater.

Device: Any entity containing an implementation of the ONE-NET protocol.

Unit: A logical, addressable subset of a Device. A Unit is primarily an application-layer construct, but is understood by some MAC-layer elements (i.e., admin messages). Valid Unit addresses are 0x00–0x0E; address 0xF is reserved, and used to refer to a specified Device as a whole (e.g., to determine power status).

Unit Type: The (generally) physical type of a Unit. Examples of Unit Types are Switch, Relay, Humidity Sensor, Temperature Sensor, Motion Sensor.

Unit Status: This will vary greatly between Unit Types. For example, the Status of a Switch will be On or Off.

Status Update: A Single Message sent from one Device / Unit to another informing the other Device / Unit of this Device / Unit's Status

Peer To Peer Communication: Two Client devices communicating with each other directly without the involvement of the Master.

Peer Table: A table stored in memory that stores a list of Device ID / Unit pairings dictating which Devices and Units need Status Updates when a Unit's Status changes.

Device Table: A table stored in memory that stores the Features, Message IDs, Number of Hops between devices, and other information needed for communication between Devices.

Fragment: The data payload in a single Block or Stream data packet.

Segment: Same as Fragment.

Chunk: One or more Segments in a Block Transaction. Block Transactions are sent one Chunk at a time before the sender waits for an acknowledgement from the recipient.

Chunk Size: The number of Segments in a Chunk of a Block Transaction.

Fragment Delay: The time that the sender of a Block or Stream Transaction should wait between sending consecutive Segments of a Block or Stream Transaction.

Segment Delay: Same as Fragment Delay.

Chunk Pause: The time that the sender of a Block Transaction should wait after receiving a response from the recipient after sending a Chunk of data. The Chunk Pause is intended to prevent the Block Transaction from monopolizing all bandwidth on a particular channel.

Features: The capabilities of a device. Examples include Boolean bits representing whether a device can serve as a multi-hop repeater, whether the device sleeps, whether the device maintains a peer table, and whether the device can handle Block and Stream Transactions.

Capabilities: Same as Features.

Keep-Alive Interval: The maximum interval in milliseconds that the Client should go without sending a packet to the Master.

Master: A Device that administers the ONE-NET channel. This Device is powered on at all times, so is not low-power (and is typically line-powered).

Network Encryption Key: A 128-bit key used to encrypt the content of Single and Block Transactions.

Replay Attack: A malicious attempt by a device which is not a valid member of a ONE-NET network to fraudulently send a message to a device inside the network by pretending to be another device in the network and re-transmitting a message that had been previously sent.

Message ID: A 12-bit value negotiated between two ONE-NET devices which serves both to delineate different transactions and as a countermeasure to Replay Attacks.

Response Timeout: The maximum amount of time that a Device waits for a response for its message.

Retransmission Delay: The amount of time (2ms for high-priority Transactions, and 10ms for low-priority Transactions) that a Device waits before retransmitting a packet, after it has timed out awaiting a response. This delay incorporates an exponential back-off component to help avoid collisions (which cannot be detected).

Single Transaction: A transfer of a small, fixed number of data bytes that fit into a single data packet, and are delivered reliably.

Stay-Awake Time: The amount of time for which a Device must stay awake and listen for packets after it has been requested to stay awake.

Stream Transaction: A transfer of an undetermined number of data bytes, which are not delivered reliably.

Message Priority: The priority that a message or transaction should proceed at. This will affect what devices need to wait when the network is congested.

Priority: Same as Message Priority.

2.2 Acronyms / Abbreviations

ACK	Acknowledge
CRC	Cyclic Redundancy Check
CSMA	Carrier Sense, Multiple Access
CTS	Clear to Send
DID	Device ID
DST	Destination
FSK	Frequency Shift Keying
FTRS	Features
LRU	Least Recently Used
MSB	Most Significant Byte
NACK	Negative Acknowledge
NID	Network ID
P2P	Peer to Peer
PCON	Packet Content
PTYP	Packet Type
RCVR	Receiver**Error! Bookmark not defined.**
RF	Radio Frequency
RPTR	Repeater
SID	System ID
SNDR	Sender
SOF	Start of Frame
SRC	Source
UID	Unique ID
XTEA	Extended Tiny Encryption Algorithm

Figure 2.1 ONE-NET Acronyms

3 Protocol Structure

The ONE-NET protocol is optimized to reduce Client power needs, since these are typically low-power, low-bandwidth devices. It is based on a star topology, with the Master having the ability to assign Clients for P2P communication. Access to the channel is via CSMA.

All data to be transmitted are encoded using a 6-bit to 8-bit mapping to maintain good clock synchronization and DC balance in the receiver.

3.1 Topologies

- Star
- Peer-to-Peer
- Multi-Hop

3.1.1 Star Topology

The Master is the center of a ONE-NET network. Any time a Device is added, it is assigned an address and other network parameters by the Master. The Master coordinates encryption key management, sets up Clients for direct (P2P) communication, and is able to act as a router for Clients.

3.1.2 Peer-to-Peer Topology

P2P is set up via a Master, based on a Client's application and bindings[1]. Once the Master has defined two or more Units as peers, these can communicate directly with one another, even if the Master is removed from the network. A *receiving* peer does not need to know that it is part of a P2P connection set up by the Master. It simply responds to the peer Unit that addressed it; therefore, a receiving Device can be part of many P2P connections. For *sending* purposes, however, each Device must store its peer table in memory.

Even though a P2P association has been set up, it may be desirable for a Master to know when application-level status (e.g., temperature, switch state) for a Client has changed. Each Client must maintain an internal flag that the Master sets if the

[1] Bindings are assigned relationships (peer-to-peer connections) between Units. The times at which Units are bound is implementation dependent, and outside the scope of this document.

Master wants to know when such status has changed. If a Client has this flag set, it will notify the Master of all status changes.

When two devices first communicate, at least two things must be exchanged before any actual data is exchanged. One, each device must send the other device its Features. Two, the devices must negotiate an agreed-upon initial Message ID. Thus, whenever two devices first communicate with each other, at least two messages are usually passed before the actual data can be transferred. To increase efficiency, a table storing this data is maintained so that the Features and Message IDs do not need to be re-negotiated every time two devices communicate. The size of the Master's table is always large enough so that a Client device never "falls off" the table. In addition, all Client devices maintain the Master device's Message ID and Features. For Client-to-Client communication, the table will be large enough to keep track of a certain number of other Client devices. If this table is not large enough to store all Clients, a Least-Recently-Used scheme is used to determine which device "falls off" the table. If a device "falls off" the table, the two devices will need to "re-sync" their Features and Message IDs the next time they communicate.

3.1.3 Multi-Hop Topology

ONE-NET also contains optional support for multi-hop networking. To accomplish Multi-Hop networking, Client Repeaters are needed. Multi-Hop packets are identified by their Packet Types (PTYPs), so as not to be confused with non-Multi-Hop packets. In this manner, Client Repeaters can repeat packets that need to be repeated without wasting time and bandwidth repeating packets that can be heard by their intended recipients without the aid of a repeater.

Multi-Hop packets contain a 3-bit Hops field, which gives them a fixed lifetime, preventing them from propagating indefinitely within the network. Multi-Hop packets also contain a 3-bit Maximum-Hops field, specifying the greatest number of hops allowed for the packet. This enables the recipient of a Multi-Hop packet to know how many hops the ACK/NACK will need to reach the sender. The originator of a multi-hop packet initially sets the Hops field to 0 and the Maximum-Hops fields to the maximum number of hops that should be taken. When a Client Repeater receives a Multi-Hop packet, it increases the Hops value. If that value is less than the maximum number of hops, it re-transmits the packet. When a Multi-Hop packet is received by its intended recipient, it looks at the Hops field to determine how many hops it took to deliver the message. The receiving Device can then improve efficiency by using that value in the maximum-hops and remaining-hops fields of the ACK/NACK reply to the sender.

Multi-Hop support in ONE-NET is optional, because the range enjoyed by ONE-NET Devices is such that most applications in the home/small business environment will not have a need for it. When a ONE-NET Device sends a message to another Device, it first attempts to do so normally. If that Transaction fails, the Device has the option of re-sending the message as a multi-hop packet, which will reach its destination if there is a sufficient number of Client Repeaters allowing for successfully propagation to the intended recipient. In those rare cases when multi-hop networking is required, a few well-placed Client Repeaters are normally sufficient to provide an effective outdoor range of several kilometers (typically used for neighborhood or agricultural applications).

Note that each time a device is added or removed from the network, the other devices are notified by the message. Part of the information in the notification message is the number of Multi-Hop-capable and Multi-Hop-Client-Repeater-capable devices in the system. Multi-Hop communication should not be attempted if there are no repeaters in the system. The number of repeaters in the system should also be considered when deciding the maximum number of hops for the message.

3.2 Quality of Service

Two levels of service exist, each a function of the application type. Generally, user interface actions (such as pushing a button and waiting for something to happen) are treated as high-priority Transactions, though this is application-specific. All other Transactions are considered low priority. High-priority Transactions are given greater opportunity to gain access to the channel by using a shorter back-off time and going directly to the front of the outgoing message queue. Note that the Response Timeout value is the same for both priority levels. See Section 13 for more information on ONE-NET message priorities.

3.3 Channel Access

Channel access is via CSMA. If the sending Device detects that the channel is busy, it waits for a Channel Back-Off Time period before checking the channel again.

Because collisions cannot easily be detected, lost packets are treated as if they collided. The Response Timeout used to detect whether or not a packet should be retransmitted is 50ms, regardless of priority.

An exponential back-off timer determines when to retransmit the packet (i.e., the length of the retransmission delay); this timer is always set to a randomly computed value between zero and some upper bound. The initial upper bound is 2ms for high-priority packets and 10ms for low-priority packets. With each subsequent retransmission, the current upper bound is doubled. Eight attempts are made to retransmit a packet before the Transaction is considered to have failed. Only Single and Block Transfer data packets are retransmitted.

To ensure that all Devices receive opportunities to use the channel, a sending Device must always release the channel after sending a packet, and wait a period equal to the Channel Back-Off Time before trying to access the channel again. The delay is measured from the end of packet transmission.

Block and Stream Transactions additionally incorporate a Fragment Delay, which defines the minimum amount of time between data packets. Default values for the Segment Delays are 25 ms for high-priority Transactions, and 125ms for low-priority Transactions, although these can be adjusted by the Master to manage traffic density. In all cases, the Master must ensure that (1) the Segment Delay for high-priority Transactions is shorter than that for low-priority ones, and (2)

Devices intended to communicate directly with one another are using the same Segment Delays.

3.4 Addressing

3.4.1 Network ID

The NID is a (globally) unique 36-bit value given to a Master Device during manufacturing. When a Master invites a Client to join the network, the Master passes this unique value to the Client.

3.4.2 Device ID

The DID is a 12-bit value, unique within the network, that is assigned to a Client by the Master when the former joins the network. The DID 0x000 is reserved for use as a broadcast address (e.g., for a Master Invite New Client). The Master's DID is always 0x001. All client devices will be assigned a DID between 0x002 and 0xFFF, inclusive.

3.4.3 System ID

A ONE-NET System-ID (SID) address is comprised of 48 bits. These 48 bits consist of a 36-bit Network ID (NID) and a 12 bit Device ID (DID). This is a shorthand method to refer to a single device on the network within the source code.

36-bit NID	12-bit DID

Figure 3.1 SID

4 Packet Structure

ONE-NET packets range in size from 240 to 504 bits, depending on the packet type. Each packet contains a header that consists of Preamble, Start-Of-Frame indicator (SOF), Repeater Device DID (RPTR DID), Message CRC, Destination Device ID (DST DID), Network ID (NID), Source Device ID (SRC DID), and Packet Type (PTYP) fields. This header is followed by a Packet Content (PCON) field, which contains data specific to the packet type. *Note that the leading seven fields, shaded in* Figure 4.1 – General Packet Format for Non-Multi-Hop Packets, *are elided (…) in subsequent diagrams. They are the same for all packets and contain 152 bits / 19 bytes.*

Preamble PATTERN 3 BYTES 0x555555	SOF PATTERN 1 BYTE 0x33	RPTR DID RAW – 12 BITS ENCODED – 16 BITS	MSG CRC RAW – 6 BITS ENCODED – 8 BITS	DST DID RAW – 12 BITS ENCODED – 16 BITS	NID RAW – 36 BITS ENCODED – 48 BITS	SRC DID RAW – 12 BITS ENCODED – 16 BITS	PTYP RAW – 12 BITS ENCODED – 16 BITS	PCON RAW – Varies ENCODED – Varies

Figure 4.1 – General Packet Format for Non-Multi-Hop Packets

Multi-Hop packets are identical to their non-multi-hop counterparts with the exception that each multi-hop packet has a Hops field at the end of the packet. See *3.1.3 Multi-Hop Topology* for more information on multi-hop.

…	PTYP RAW – 12 BITS ENCODED – 16 BITS	PCON (optional, depending on PTYP) RAW – N BITS ENCODED – M BITS	HOPS RAW – 6 BITS ENCODED – 8 BITS

Figure 4.2 – General Packet Format for Multi-Hop Packets

4.1 Data Format

All data are sent MSB. Each individual byte is sent most significant bit first.

Note that there are, in effect, four "stages" through which data passes while being packetized and prepared for transmission. These can be considered as follows:

Raw Data — data as viewed by applications

CRC-Applied Data — a segment of raw data to which has been attached CRC information (to provide accuracy)

Encrypted Data — CRC-applied data to which an encryption algorithm has been applied (to provide confidentiality)

Encoded Data — data that has been converted to a form that more evenly balances the distribution of bits (to simplify receiver design, and increase accuracy)

4.2 Packet Elements Common To All Packets

Please see Figure 4.1. This section breaks down each element within Figure 4.1.

4.2.1 Preamble And Start-Of-Frame

This is a four byte constant (0x55555533). The first four bytes of all ONE-NET packets is 0x55555533. These four bytes were selected because they have equal numbers of 0's and 1's. Such a pattern is needed by the transceiver to detect the start of a packet. Each transceiver type has different requirements, but all or almost all require a bit pattern to be loaded into the transceiver's registers. Anyone porting ONE-NET to a new transceiver will need to load these bytes into the transceiver's registers.

4.2.2 Repeater DID

This is the DID of the device actually sending a ONE-NET packet from its transceiver. For non-multi-hop packets, the Repeater DID and the Source DID will be the same. For multi-hop packets, the repeater DID and the source DID will be the same on the first hop, but different on subsequent hops. A Multi-Hop-Repeater device will replace the Repeater DID of the packet it receives with its own DID, adjust the number of hops, and then resend the packet.

4.2.3 Message CRC

This is a CRC calculated over all fields starting with the Destination DID through the end of the PCON. For non-multi-hop packets, this means that the CRC is calculated from the Destination DID field to the end of the packet. For multi-hop packets, this means that the CRC is calculated from the Destination DID field up to, but not including, the Hops Field. The calculated CRC is an 8-bit CRC (see section 4.4.2 (DS—check) for more details on how ONE-NET CRCs are calculated). The left-most 6 bits are retained and encoded and placed into the Message CRC field.

4.2.3.1 Message CRC Example (see also sections 5.3 and 5.4)

Assume the following 30-byte non-multi-hop ONE-NET packet…

55 55 55 33 B4 BC XX B4 B3 B4 B4 B4 B4 B4 BC B4 BC B5 B3 CA CA A3 56 B5 9C 9A D5 D9 6A C9

The six blue bytes represent the preamble and repeater DID. These bytes are not part of the Message CRC calculation. The next byte (XX) is the Message CRC field. It is marked as XX because we need to calculate the Message CRC in order to fill that value in. The red bytes are the bytes that make up the calculation of the Message CRC.

1. The computed 8-bit CRC over the red bytes is 0xFC (see section 4.4.2 (DS_CHECK) for more information on how the 8-bit CRC is computed).
2. 0xFC corresponds to 11111100 in bits.
3. The two rightmost bits are thrown away, so we are left with 111111, which is 0x3F in hex.
4. 0x3F is then encoded using ONE-NET's 6-to-8-bit encoding scheme. The result of the encoding is 0xD2.
5. 0xD2 is placed in the Message CRC field.

55 55 55 33 B4 BC D2 B4 B3 B4 B4 B4 B4 B4 BC B4 BC B5 B3 CA CA A3 56 B5 9C 9A D5 D9 6A C9

The message above is sent. When the message is received, the process is repeated again by the device receiving the message. The expected Message CRC over the red bytes is determined as 0xD2. The recipient checks to make sure that the Message CRC byte is actually 0xD2. If it is not, the message is rejected.

In addition to the Message CRC, there is also a Payload CRC that will be discussed later. The Payload CRCs much also match for the packet to be treated as a valid packet.

The Message CRC bytes were chosen intentionally to not include either the Repeater DID or the Hops byte (irrelevant for the non-multi-hop packet above). The purpose of this is so that a Repeater DID does not have to calculate or change the Message CRC when repeating the packet. Since the Repeater DID changes both the Repeater DID and Hops fields, if they were part of the Message CRC calculation, the Message CRC would have to recalculated and changed by the repeater.

4.2.4 Destination DID

This is the intended recipient of the message. It will never be the same as either the Source DID or the Repeater DID. The Destination DID will be an individual physical device or it will be "0xB4B4", which decodes to device 0. A Destination DID of 0 (which no Device will have) is considered a Broadcast message. Devices should ignore messages that are neither to them specifically and are not Broadcast messages with the following exception. Client devices serving as Multi-Hop Repeaters should repeat multi-hop packets not addressed to them assuming there are any hops remaining in the packet and assuming they are not currently in the middle of a transaction themselves.

4.2.5 Network ID (NID)

This is a 36-bit decoded / 48-bit encoded value. All devices in a network will share this value. No devices outside of the network will have this value. This value is given to a client device by the master when the client device joins a network. All devices which are currently in a network should ignore all messages where the NID does not match their own network DID. The only device that should respond to a message that does not match its NID is a client that is not currently in a network and is trying to join. In that one instance, the client should respond to "Invite" packets from a master that appear to be addressed to them.

4.2.6 Source DID

This is the device that is originally sending a message or a response. For non-multi-hop messages and in multi-hop messages where the number of hops is 0, the Source DID and the Repeater DID will be the same. For "Repeated" messages, this value will be different. The Destination Device should respond to the Source Device if a response is required.

4.2.7 Packet Type (AKA PTYP or PID)

The packet type, also known as the PTYP or PID, is a 16-bit encoded, 12-bit decoded field. The twelve decoded bits should be interpreted as follows. The multi-hop and stay-awake bits should be interpreted as Boolean values where 1 means true and 0 means false.

# of Payload Blocks 4 BITS	Multi-Hop Bit 1 BIT	Stay-Awake Bit 1 BIT	Packet Message Type 6 BITS

Figure 4.3 PID Bytes

4.2.7.1 Packet Message Types

The Packet Message Type is a value from 0x00 to 0x3F, specified in the six least significant bits of the PTYP. As of Version 2.3.0, only values 0x00 to 0x0F are defined, and not all of these values are used. There are four main families of packets.

1. Single Packets – All information is contained within one packet. "Single Data", "Single Data ACK", and "Single Data NACK" packets are examples.
2. Block Data Packets – Packets where all then information CANNOT fit in one packet, but instead is contained in a SERIES of packets. Block packets will be re-sent if they are not acknowledged.
3. Stream Data Packets – Packets where all then information CANNOT fit in one packet, but instead is contained in a SERIES of packets. Stream packets will NOT be re-sent if they are not acknowledged, thus they are considered "unreliable".
4. Invite Packets – Packets sent by the master inviting a device to join the network using a shared encryption key known only to the master and the device being invited.

4.2.7.1.1 Acknowledgement Packets

Acknowledgment packets are sent to acknowledge the receipt of a data packet. Acknowledgement packets can be either "Positive Acknowledgements" (ACKs) or "Negative Acknowledgements" (NACKs). Acknowledgement packets are never the first packet sent in a transaction. They are only sent it response to other packets. An "ACK" is interpreted as "I received your message and everything is OK". A NACK is interpreted as "I received as your message and there is a problem."

4.2.7.1.2 Data Packets

This definition can be confusing since Acknowledgement Packets can contain data. The best way to think of a Data Packet is any packet that isn't an Acknowledgement Packet or an Invite Packet. Another way to think of it is a packet that requires an Acknowledgement Packet. Data Packets can be the first packet in a transaction. Acknowledgment Packets cannot.

Packet Message TYpes	Raw Hex	Encoded Hex
Single Data	0x00	0xB4
Single Data ACK	0x01	0xBC
Single Data NACK	0x02	0xB3
Route	0x03	0xBA
Route ACK	0x04	0xB5
Route NACK	0x05	0xB9
Block Data	0x06	0xB6
Block Data ACK (unused as of 2.3.0)*	0x07	0xB2
Block Data NACK (unused as of 2.3.0)*	0x08	0xC4
Block Terminate (unused as of 2.23.0)*	0x09	0xCC
Stream Data	0x0A	0xC3
Stream Data ACK (unused as of 2.3.0)*	0x0B	0xCA
Stream Data NACK (unused as of 2.3.0)*	0x0C	0xC5
Stream Data Terminate (unused as of 2.3.0)*	0x0D	0xC9
Master Invite New Client	0x0E	0xC6
Client Request Invite (unused as of 2.3.0)*	0x0F	0xC2
• As of Version 2.3.0, Block and Stream ACK, NACK, and Terminate Messages are transmitted as Single Data, Single Data ACK, or Single Data NACK messages. These codes are reserved for future ONE-NET versions in case they are desired. • Client Request Invite is not implemented as of Version 2.3.0.		

Figure 4.4 Packet Message Types

4.2.7.2 *Example*

The following is an example of the parsing the two bytes of an Encoded PID.

Encoded PID : 0x34C3

Decoded Bytes : 0x40A : 12 Raw Bits : 010000001010

# of Payload Blocks	Multi-Hop Bit	Stay-Awake Bit	Type
0100 = 4	0	0	001010 = 0x0A

Figure 4.5 PID Bytes Example

The number of payload blocks in the packet is 4. The packet is not a multi-hop packet. The packet is not a stay-awake packet. The type of the packet is type 0x0A, which corresponds to "ONE-NET Raw Stream Data".

4.2.7.3 *Determining The Packet Length From The PID*

The total number of bytes in a ONE-NET packet is based on two factors.

1. The number of payload blocks.
2. Whether the packet is a multi-hop packet.

Both of these factors can be determined from the Decoded PID. Each payload block is eight bytes (64 bits). In addition, each ONE-NET packet has a two-bit encryption technique. Valid ONE-NET packets must have from 1 to 4 payload blocks, inclusive, so the number of payload bits will be 64, 128, 192, or 256. These bytes are encrypted. In addition, ONE-NET packets have two unencrypted bytes representing the encryption technique. Hence the number of unencoded payload bits with technique will be 66, 130, 194, or 258.

All ONE-NET packets go through a process of encoding all bits, which involves a 6 to 8 bit transformation, so the number of encoded bits required can be determined by multiplying the number of decoded bits needed by 8, then dividing by 6.

Fractional results need to be rounded up. In addition, since ONE-NET uses eight bits per byte, any result that is not a multiple of 8 needs to rounded up. Hence the number of encoded bits required is as follows…

# Of Blocks	# Of Decoded Encrypted Bits	# Of Decoded Bits Including 2-Bit Encryption Technique	# Of Encoded Bits (6 To 8 Transform)	# Of Encoded Bits (Multiple Of 8)	# Of Encoded Payload Bytes With Technique
1	64	66	88	88	11
2	128	130	173.33	176	22
3	192	194	258.67	264	33
4	256	258	344	344	43

Figure 4.6 Payload Sizes

From the number of encoded payload bytes including the encryption technique, we can add the 19 encoded bytes that are precede the payload bytes and are common to all ONE-NET packets. In addition, for multi-hop packets, there is an additional byte AFTER the encoded payload bytes…

# Of Blocks	# Of Encoded Payload Bytes With Technique	# Of Bytes Preceding Payload Bytes	Number Of Bytes For Non-Multi-Hop Packet	Number Of Bytes For Multi-Hop Packet
1	11	19	30	31
2	22	19	41	42
3	33	19	52	53
4	43	19	62	63

Figure 4.7 Packet Sizes

4.2.8 Packet Contents (PCON)

Although the Packet Content (PCON) field differs considerably among the various packet types, there are some commonalities.

1. All PCON payloads are encrypted using XTEA encryption.
2. All PCON payloads contain an 8-byte Payload CRC as the first byte both for security purposes and for error detection.
3. All PCONs contain a two-bit unencrypted field which represents the encryption technique.
4. All payloads except stream packet payloads use 32 rounds of encryption.
5. Stream packet payloads use 8 rounds of encryption.
6. All packets except for invite packets contain a message ID.

Packet types can be roughly broken into four categories (see also section 4.2.7.1)…

1. Invite Packets – These are packets sent from the master to a client inviting the client to join the network.
2. Data Packets – These are packets from one device to another containing data of some sort.
3. Response Packets – These are packets that are responses to data packets. These responses may contain data themselves.
4. Route Packets – These are packets designed to establish a route from one device to another.

Further divisions within packet types can be made. As discussed in section 4.2.7, all packets are either “Stay-Awake” or “Multi-Hop” packets. That information can be determined from the PTYP bytes. See section 4.2.7 for more details.

1. Invite Packets
    a. Currently there is only one type of invite packet. A master will send an invitation to a client that has not joined the network yet. That message will be encrypted using an encryption key known to the master and the device waiting to join.
    b. In the future, there may be an option for a client to initiate the invitation process. As of ONE-NET Version 2.3.0, this has not been implemented. However, a PTYP code has been reserved for implementation in the future.
2. Data Packets – Data Packets can be broken into three categories.
    a. Single Data Packets – All the information to be sent is contained in one packet. Single Data Packets can be broken into four categories.
        i. Application Messages – These are packets which transmit information that can only be handled by the application code. Examples would be commands to turn a light on or off or status messages reporting power usage.
        ii. Administration Messages – These are messages handled by the ONE-NET code generally. Examples include, but are not limited to…
            1. Key changes
            2. Fragment Delay Changes
            3. Channel or Data Rate Changes
            4. All sorts of other network settings
        iii. Features Messages – These messages contain the features bytes of a device. The features bytes can be thought of as a device’s capabilities. For example, if device A sends device B its features bytes, device B will know, among other things, whether device A (knowing these things will allow device A…
            1. Can store peer devices and units in a table.
            2. Can function as a multi-hop repeater.
            3. Can participate in Block or Stream Messaging

        4. Ever goes to sleep'

    iv. Route Messages – These messages are sent to determine what devices are in range of what other devices and to determine the expected round-trip time of a message.

3. Response Packets – Response Packets can roughly be divided into two categories…

    a. Positive Acknowledgements (ACK messages) – The message was received and everything is OK. Within positive acknowledgements, the receiving device may or may not send back a payload with the ACK. Payloads might include a data, feature, or route message.

    b. Negative Acknowledgements (NACK messages) – The message was received, but something is wrong. NACK messages should be accompanied by a NACK reason, whether the problem is a temporary problem, and how the problem can be fixed.

        i. An example of a message that cannot be fixed would be a Command to turn on a relay on a certain unit. If that target unit either does not exist or if that unit exists, but is not a relay, the command cannot be honored. The receiving device will send back a "fatal" NACK reason explaining what went wrong. A "fatal" NACK reason is sent when a device cannot honor a request and never will be able to honor that request.

        ii. An example of a message that can be fixed would be the same Command to turn on a relay on a certain unit. In this case, the unit does exist and is a relay. However, there could still be a problem. Some problems could be…

            1. The receiving device needs to know the sending device's features. The NACK reason will specify that it wants the sending device to send its features and try again.

            2. The sending device used an invalid Message ID. In this case, the receiving device should NACK and respond with a Message ID that it will accept.

            3. The receiving device, for whatever reason, cannot honor the command at this particular time, but will honor the request in the future. In this case, the receiving device will NACK, but within the NACK will tell the sending

device that it can try again. It will possibly give a certain time to wait before trying again.

4. Route Packets
    a. A route packet determines a route between two devices. The route packet starts with the source device attaching its DID. The best way to describe a route packet is to imagine the following example…
        i. Device 003 wants to determine the best route to device 006.
        ii. Device 003 is in range of 004 and vice-versa. 004 is a repeater.
        iii. Device 004 is in range of 003 and 005. 005 is a repeater.
        iv. Device 005 is in range of 006.
    b. Device 003 initiates the message by filling the payload with 0's and then attaches its own DID to the front. So the message payload is sent as 0x003. The source is 0x003, the destination is 0x006.
    c. Device 004 receives the message, notes that it is not the recipient. It also checks whether 006 is in the message. If 006 is in the message is on its way back. If not, 006 has not yet seen the message. 004 now checks to see if it has already seen the message in the current leg of the trip. If so, it will ignore it. However, in this case, it has not, so it attaches its DID and sends it along. The message is now 003004.
    d. Device 005 receives the message, notes that it is not the recipient. It also checks whether 006 is in the message. If 006 is in the message is on its way back. If not, 006 has not yet seen the message. 005 now checks to see if it has already seen the message in the current leg of the trip. If so, it will ignore it. However, in this case, it has not, so it attaches its DID and sends it along. The message is now 003004005.
    e. Device 006 receives the message and notes that it is the recipient. It also checks whether 006 is already in the message. If 006 is in the message it will ignore this message since it has already responded. If not, it will attach itself and send the message back. In this case, it has not, so it attaches its DID and sends it along. The message is now 003004005006.
    f. Device 005 receives the message, notes that it is not the recipient. It also checks whether 006 is in the message. If 006 is in the message is on its way back. 006 is in the message, so it on the return trip. 005 now checks to see if it has already seen the message in the current leg

of the trip. If so, it will ignore it. It notices that it is in the message already, but only on the first leg of the trip,. Not the return leg, so it attaches its DID and sends it along. The message is now 003004005006005.

g. Device 004 receives the message, notes that it is not the recipient. It also checks whether 006 is in the message. If 006 is in the message is on its way back. 006 is in the message, so it on the return trip. 004 now checks to see if it has already seen the message in the current leg of the trip. If so, it will ignore it. It notices that it is in the message already, but only on the first leg of the trip,. Not the return leg, so it attaches its DID and sends it along. The message is now 003004005006005004.

h. Device 003 sees the message and notes that it was the source. Therefore a route has successfully been established. From this route, 003 knows the following…

    i. It can successfully transmit to device 006.

    ii. It has a good idea of how long to expect to get a response since it kept track of when the original message was sent. This will help it decide how long it needs to wait before deciding that the message was lost.

    iii. It knows that it takes two hops to get a message to 006.

    iv. It knows that in order to get a message to 006, it needs to have 004 and 005 available as repeaters.

i. From this information, it can plan accordingly when sending messages to 006. In particular, for block and stream messages, it will know that it needs to request permission to reserve devices 004 and 005 as repeaters when communicating with device 006.

4.2.9 PCON Structure

Please see Section 4.2.8 for more information. The PCON field will depend on the packet type. The Payload CRC field is common to all packet types. Please see Section 5 for more details on how the Payload CRC is calculated. All data except for the encryption method is encrypted. Currently all stream packets use 8 rounds of XTEA encryption. All other packets use 32 rounds. The encryption method field specifies this. If the encryption method does not match, the packet is rejected.

4.2.9.1 Single Data Packets

All single data packets have the following fields…

Pld. CRC 8 BITS	Message ID 12 BITS	Message Type 4 BITS	Transaction Data Payload 40, 104, or 168 BITS	Encryption Method 2 BITS

Figure 4.8 – PCON Structure for Single Data Packets

4.2.9.1.1 Payload CRC

The Payload CRC is calculated over the Message ID, Message Type, and Data Transaction Fields. If the calculated payload CRC does not match the payload CRC in the packet, the packet is rejected. Please see section 5 for more details on how CRCs are calculated in ONE-NET.

4.2.9.1.2 Message ID

The Message ID helps both to prevent duplicate messages and provides security against replay attacks. The message ID is a 12-bit value agreed upon by both devices.

When receiving a message from a device, the sending device and the Message ID will fall into one of five categories…

1. The sending device is not a device that this device has communicated with before. In this case, the sending device will not be in this device’s table. Thus an acceptable message ID will be randomly generated, the message will be NACK’d, and the sending device will be told to try again with the acceptable message ID.
2. The sending device is already in the recipient device’s table, but the Message ID is LESS THAN the one expected. The recipient device will NACK the message and send back an acceptable Message ID to use. This is done to prevent replay attacks.
3. The sending device is already in the recipient device’s table and the Message ID is exactly the Message ID that is expected. In this case, the message will be ACK’d. However, since the Message ID is exactly the same as the one that was expected, the message will not be acted upon since it is assumed that the message has ALREADY been acted upon and the sending device simply did not receive the acknowledgement.
4. The sending device receives a Message ID GREATER THAN what it has on its table. In this case, that means that the message is a new message. It will act upon and ACK that message and store the new message ID in its table.
5. The Message ID is acceptable, but it is close to 4095, which is the maximum Message ID. To avoid future replay attacks, the receiving device will request that the master change the network key. Once the key change is made, the Message IDs will be set to a very low random number and thje two devices will re-sync Message IDs as in case 1.

All of this assumes that everything else in the message is acceptable. If the message needs to be NACK’d for some other reason, the new Message ID will not be accepted and the message will not be acted upon. The goal is to ACK only legitimate messages and to do act upon them exactly once (i.e. if told to toggle a relay, if the message is acted upon twice, doing so would cause a rapid locking, then unlocking of a door, which is likely not what the sending device intended.

Please see section 6 of the **Preventing Replay Attacks And Other Security Considerations In ONE-NET** document for more information on how ONE-NET uses Message IDs to prevent replay attacks.

4.2.9.1.3 Message Type

This is a 4-bit field which determines how the transaction data payload should be interpreted. Options are, as discussed previously…

1. Application Messages – explained more thoroughly in Section 5.
2. Admin Messages – explained more thoroughly in Section 6
3. Features Messages – explained more thoroughly in Section 7.
4. Route Messages – explained more thoroughly in Section 8.

All features messages have 40-bit payloads and only the first 32-bits are relevant. Hence the number of XTEA Blocks in the PTYP should be 1. If the number of XTEA Blocks in the PTYP is NOT 1, the message should be rejected. All route messages have 168-bit payloads. Hence the number of XTEA Blocks in the PTYP should be 3. If the number of XTEA Blocks in the PTYP is NOT 3, the message should be rejected. Application and Admin payload lengths may be 40, 104 or 168 bits. The number of relevant bits will vary. If the number of XTEA Blocks in the PTYP is NOT 1, 2, or 3, the message should be rejected. In addition, if the number of XTEA blocks is too large for the relevant data, the packet should be rejected.

In addition, ONE-NET defines values named ON_RESERVED_MSG_TYPE_1, ON_RESERVED_MSG_TYPE_2, and ON_RESERVED_MSG_TYPE_3 for future use. As of ONE-NET Version 2.3.0, these message types are undefined and should therefore be considered invalid. They correspond to values 7 to 9. Values 10 to 15 are available as application-specific parsing codes. ONE-NET allows applications to define there own parsing mechanisms if none of the others fit. If none of the other message types fit, application designers can use one of these values and write their own code to parse / interpret the payload data. Please be aware that such application-specific parsing mechanisms will not be portable to other applications. Hence, for portability purposes, whenever possible, one of the ONE-NET defined parsing mechanisms should be used.

4.2.9.1.3.1 Application Message Types (0x00 to 0x03)

Please see section 5 for more information. There are four pre-set parsing mechanisms for application messages. An application message is generally anything that does not fall into the category of an administration, route, or features message, and which requires application-specific code to implement. Examples of application messages would be the following…

1. A command to turn a switch on or off.
2. A command to set the volume of a speaker to a certain level.
3. A request for the current humidity level that a humidity sensor is reading.

Types of Application Messages

1. ON_APP_MSG Message Type = 0x00
    a. 40-bit payload message data should be interpreted as the following…
        i. 4 bit message class
        ii. 8 bit message type
        iii. 4 bit source unit
        iv. 4 bit destination unit
        v. 20 bit signed integer value
2. ON_APP_MSG_TYPE_2 Message Type = 0x01 – same as ON_APP_MSG, but source unit and destination unit are missing and the signed integer value is 28 bits, not 20.
    a. 40-bit payload message data should be interpreted as the following…
        i. 4 bit message class
        ii. 8 bit message type
        iii. 28 bit signed integer value
3. ON_APP_MSG_TYPE_3 Message Type = 0x02
    a. 40-, 104-, or 168-bit payload message data should be interpreted as the following…
        i. 8 bit message type
        ii. The rest of the data will be parsed based on the message type. Generally this message type is reserved for "special" types of

ONE-NET message types which do not fall neatly into a single signed integer value. Examples are date and color messages.

4. ON_APP_MSG_TYPE_4 Message Type = 0x03
    a. 40-, 104-, or 168-bit payload message data should be interpreted as an array of bytes. The interpretation of the data is generally left to the application code.

4.2.9.1.3.2 Admin Message Type = 0x04

Please see section 6 for more information. Administration message type payloads are parsed as follows…

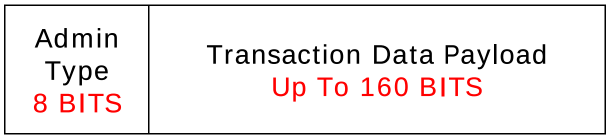

Figure 4.9 – Administration Message Parsing

The number of relevant bytes and the manner in which the bytes will depend on the Admin Type. The Admin Type codes are defined as follows…

- ON_FEATURES_QUERY = 0x00
    - Sent to request another device's features. Payload is the requesting device's features (4 bytes total).
- ON_FEATURES_RESP = 0x01
    - Payload is the device's features (4 bytes total).
- ON_NEW_KEY_FRAGMENT = 0x02
    - Payload is a key fragment (4 bytes total).
- ON_ADD_DEV_RESP = 0x03
    - Response to acknowledge that the device has been informed that a device has been added. No payload.
- ON_REMOVE_DEV_RESP = 0x04
    - Response to acknowledge that the device has been informed that a device has been removed. No payload.
- ON_CHANGE_DATA_RATE_CHANNEL = 0x05

    - Payload is one byte for the new channel, one byte for the new data rate, two bytes for the number of milliseconds to wait before changing, two bytes for the number of milliseconds to wait for a message before reverting to the base data rate and channel (6 bytes total).
- ON_REQUEST_KEY_CHANGE = 0x06
    - Message sent to inform the master that the device would like the network to change keys. No payload.
- ON_CHANGE_FRAGMENT_DELAY = 0x07
    - Message sent to inform a device of the high and low priority fragment delays. Payload is two 16 bit values representing the high and low priority fragment delays in milliseconds (4 bytes total).
- ON_CHANGE_FRAGMENT_DELAY_RESP = 0x08
    - Message sent acknowledging that a device has received its new fragment delays. No payload.
- ON_CHANGE_KEEP_ALIVE = 0x09
    - Message sent to change a device's keep-alive interval in milliseconds. The keep-alive interval is how often the device must check in with the master (4 bytes total).
- ON_ASSIGN_PEER = 0x0A
    - Message sent by the master to assign a peer to a unit. See the document **ONE-NET Peer Messaging** for more information on peer management in ONE-NET (4 bytes total).
- ON_UNASSIGN_PEER = 0x0B
    - Message sent by the master to unassign a peer from a unit. See the document **ONE-NET Peer Messaging** for more information on peer management in ONE-NET (4 bytes total).
- ON_KEEP_ALIVE_QUERY = 0x0C
    - Message sent by the master to the client informing it of its keep-alive time and requesting that the client check-in immediately. Payload is the time in milliseconds (4 bytes total).
- ON_KEEP_ALIVE_RESP = 0x0D
    - Message sent by a client to check in with the master. The payload is

the last four bytes of the current key. If the client has the correct key, the message is ACK'd by the master. If the client has an incorrect key, the message is NACK'd and a new key fragment is given in the response. If the client receives a NACK with a new key, it changes keys and immediately checks in again (4 bytes total).

- ON_CHANGE_SETTINGS = 0x0E
  - Message sent by the master to inform the client of its new settings (1 byte total).
- ON_CHANGE_SETTINGS_RESP = 0x0F
  - Message sent by the client to confirm that it has received its new settings. No payload.
- ON_REQUEST_BLOCK_STREAM = 0x10
  - Used to request permission for a block or stream transfer. Payload includes flags, number of hops, priority, transfer size, chunk size, fragment delays, chunk pause time, channel, data rate, timeout time, and destination DID (21 bytes total).
- ON_REQUEST_REPEATER = 0x11
  - Sent to request that a device should be reserved as a repeater for a block or stream transfer. Payload includes the repeater requested, the block / stream message source and destination devices, the estimated time of the transfer, the channel and data rate to use, and the priority of the transfer (13 bytes total).
- ON_TERMINATE_BLOCK_STREAM = 0x12
  - Sent to terminate a block or stream transfer. Payload includes the status of the message (i.e. successful or not, see Section 15 for a list of Message Status Codes), the NACK reason(if any) for ending the transfer, the DID of the device terminating the transfer, and any payload attached to the ACK or NACK). (10 bytes total).
- ON_ADD_DEV = 0x13
  - Sent by the master to inform a client that a device has been added. Payload includes the device that was added and the number of multi-hop-capable and multi-hop-repeater-capable devices in the network. (4 bytes total.
- ON_ADD_DEV = 0x14

- Sent by the master to inform a client that a device has been removed. Payload includes the device that was added and the number of multi-hop-capable and multi-hop-repeater-capable devices in the network. (4 bytes total).

4.2.9.1.3.3 Features Message Type = 0x05

Please see section 6 for more information. Features messages are four bytes. They contain the four bytes of a device’s capabilities / features. These bytes are used to help other devices in the network know how to communicate with this device. For example, if the device’s features say that it is not capable of maintaining a peer table, the master will not assign it any peers. Similarly, if a device is not capable of handling multi-hop or block messages, the other devices will know not to send it those types of messages.

4.2.9.1.3.4 Route Message Type = 0x06

Please see section 7 and the “Route Packets” section of section 4.2.8 for more information. Route packets are sent to determine…

1. Whether a route exists from one device to another.
2. The number of hops that are required, if any.
3. Which repeaters are required, if any.

Route packet payloads are 168 bits and should be broken into 12-bit increments. Each 12-bit increment represents a Device ID that is part of the route, including the source and destination. 12 bits of 0’s represent the end of the route message.

4.2.9.2 Acknowledgement Packets

As mentioned, a "Positive Acknowledgement" (ACK) packet is sent when a data packet has been received and there is no problem with it. A "Negative Acknowledgement" (NACK) is sent when a packet has been received and there is a problem with it. The bytes in ACKs and NACKs are very similar to each other. The difference is that NACK packets specify a NACK Reason, which is a code for why the packet is being rejected. Therefore ACK packets have one extra byte to use since they do not need to devote a byte for the NACK Reason.

The Message ID in an ACK or NACK packet will be the same Message ID that was used in the Data packet that is being responded to. Note that the "NACK Reason" for all ACK packets is "NACK Reason No Error". In other words, there is no NACK Reason because there is no error. Since all ACK packets have this NACK Reason, it is implicit and therefore not a part of the ACK packet.

Figures 4.9.10 and 4.9.11 specify the overall byte structure of ACK and NACK packet PCONs. The NACK Reason, Handle, and Payload fields are explained in Sections 4.2.9.2.3 through 4.2.9.2.5.

Pld. CRC 8 BITS	Message ID 12 BITS	ACK Payload Handle 4 BITS	ACK Payload 40, 104, or 168 BITS	Encryption Method 2 BITS

Figure 4.10 – PCON Structure for ACK Packets

Pld. CRC 8 BITS	Message ID 12 BITS	NACK Payload Handle 4 BITS	NACK Reason 8 BITS	NACK Payload 32, 96, or 160 BITS	Encryption Method 2 BITS

Figure 4.11 – PCON Structure for NACK Packets

4.2.9.2.1 Payload CRC

The Payload CRC is the same as it is in Single Data packets. Please see Section 4.2.9.1.1 for more details.

4.2.9.2.2 Message ID

The Message ID is exactly the same as it is in Single Data packets. Please see Section 4.2.9.1.2. The Message ID in an Acknowledgement Packet will always match the Message ID that the Acknowledgement Packet is responding to. In this way, all Data packets and responses are paired. In addition to data integrity and security purposes, the Message ID field provides the receiving device the added information of knowing which Acknowledgement goes with which data packet. This can be quite valuable when two devices get out of sync. There could be a delay in messages being acknowledged, particularly with Multi-Hop packets. It's possible that a device could send two messages and get two acknowledgments, but the Acknowledgements were received out of order. The Message ID makes it clear which Acknowledgement goes with which Data message to avoid confusion.

4.2.9.2.3 Handle

The “Handle” of an ACK / NACK packet is a code for how the payload response data should be parsed. ACK and NACK payloads data might consist of any the following…

1. No payload. Some ACKs and NACKs do not require a payload.
2. A payload consisting of an unsigned integer value.
3. A payload consisting of an unsigned integer value dealing with timing.
4. A payload containing the Features of a device.
5. A payload containing an XTEA key fragment or a full XTEA key.
6. An administration message payload.
7. A status message response.
8. An array of data.

In many cases, the Data message contents will dictate the type of response that the ACK or NACK will send. In others, the payload data will only make sense when combined with the NACK Reason. For example, if a packet is NACK’d with a NACK Reason, then the expected response payload is going to be the correct key or key fragment to use. Similarly, if a packet is NACK’d with a NACK Reason of “Invalid Channel” or “Invalid Message ID” and an unsigned integer value is sent back, the value should be interpreted as the acceptable channel or Message ID to use. The interpretation of a response payload should therefore incorporate the original message, the packet size, the NACK reason, the NACK handle, and any other factors that may be pertinent. Therefore the interpretation of a packet may not be known or knowable simply by dissecting the packet.

Acceptable Acknowledgement Payload Handles are summarized in the following table.

ACK / NACK	0x00	No payload is attached to this packet
ACK / NACK Features	0x01	The 4 byte Features Payload is attached to this packet
ACK / NACK Data	0x02	An array is attached to this packet
ACK / NACK Value	0x03	A 32 bit unsigned integer value is attached to this packet
ACK / NACK Time MS	0x04	A 32 bit unsigned integer value representing a time in milliseconds
ACK / NACK Timeout Time MS	0x05	A 32 bit unsigned integer value representing the time in milliseconds that a device should wait before assuming that a timeout has occurred.
ACK / NACK Slowdown Time MS	0x06	Used for stream and block transactions to signify that the recipient wants the sender to send packets less rapidly. The attached 32-bit value is the time, in milliseconds, that the recipient wants the sender to

		slow down by.
ACK / NACK Speedup Time MS	0x07	Used for stream and block transactions to signify that the recipient wants the sender to send packets less rapidly. The attached 32-bit value is the time, in milliseconds, that the recipient wants the sender to speed up by.
ACK / NACK Pause Time MS	0x08	A 32 bit unsigned integer value representing a time in milliseconds. The recipient sends this when it is busy. It wants to continue a transaction, but it wants the sender to pause. The value is the time in milliseconds of the requested pause.
ACK / NACK Response Time MS	0x09	A 32 bit unsigned integer value representing a time in milliseconds. The recipient sends this when it wants to specify a new response timeout. It is the time, in milliseconds, that the sender should wait for a response before concluding that, if no response has been response has been received yet, the sender should assume that the packet was not received.
ACK / NACK Key or Key Fragment	0x0A	The payload is interpreted as either a 4 byte XTEA key fragment or a full 16 byte XTEA key. The packet length specified in the PID will determine which of these two interpretations is valid.
ACK Block Packets Received (ACK only)	0x0B	A bitwise array representing which packets in a chunk of a block message have been received and accepted as valid.
ACK Route (ACK only)	0x0C	An array representing a route or partial route between devices
ACK Application Message (ACK only)	0x0D	The payload should be interpreted as a Single Data Message with message type ON_APP_MSG.
ACK Admin Msg (ACK only)	0x0E	The payload should be interpreted as a Single Data Message with message type ON_ADMIN_MSG.
Application Handle (used by / interpreted only by application code)	0x0F	Not interpreted by ONE-NET. ONE-NET will treat this as array data. It is up to the Application-level code to parse and interpret the data.

Figure 4.12 Acknowledgement Payload Handles

4.2.9.2.4 NACK Reason (Relevant And Present Only For NACK Packets)

The “NACK Reason” field is only relevant for NACK packets. It consists of eight bits. NACK Reasons can be split into four categories.

1. ONE-NET-defined “non-fatal” NACK Reasons (0x00 – 0x3F). The NACK is non-fatal and all ONE-NET applications will be able to understand the reason
2. User-defined “non-fatal” NACK Reasons (0x40 – 0x7F). The NACK is non-fatal, but none of the ONE-NET-defined NACK reasons are applicable. These codes are reserved for application code writers and are therefore non-portable. They should be used, therefore, only in self-contained systems where both sides devices can interpret and understand the codes. Whenever possible, ONE-NET-defined NACK Reasons should be used.
3. ONE-NET-defined “fatal” NACK Reasons (0x80 – 0xBF). The NACK is non-fatal and all ONE-NET applications will be able to understand the reason
4. User-defined “fatal” NACK Reasons (0xC0 – 0xFF). The NACK is fatal, but none of the ONE-NET-defined NACK reasons are applicable. These codes are reserved for application code writers and are therefore non-portable. They should be used, therefore, only in self-contained systems where both sides devices can interpret and understand the codes. ONE-NET-defined NACK Reasons should be used.

4.2.9.2.4.1 “Fatal” vs. “Non-Fatal” NACK Reasons

NACK Reasons can be “fatal” or “non-fatal”. A “fatal” NACK Reason should be given when the error is non-recoverable. No matter how many attempts are made, the message will never be successful. Examples of fatal NACK Reasons are “Device Function Error” and “Unit Function Error”. The message is either addressed to a non-existent unit or is asking the device to do something it cannot do (i.e. querying a switch for a temperature status). No matter how many times the message is sent, it will never work, so a fatal NACK Reason should be given so the sending device knows not to try again. An example of a non-fatal NACK Reason would be the following…

1. The device is busy, but very soon, it will be able to handle the request. Hence the sending device should try again.
2. The sending device is using a bad channel, data rate, key, or fragment delay. In this case, the message should be rejected and the response should include the reason that it is being rejected, plus a way to repackage the request so that it will be accepted (i.e. if a fragment delay was rejected, an acceptable fragment delay should be attached). The sending device, upon receiving the NACK, will look at the payload attached and decide whether to re-package and re-send.

4.2.9.2.4.2 List Of NACK Reasons

The table below lists the NACK Reasons reserved by ONE-NET. The NACK Reason is an 8-bit value ranging from 0x00 to 0xFF. The values partition as follows…

1. 0x00 → No Error. This is an ACK.
2. 0x01 – 0x3F → ONE-NET-specified “non-fatal” NACK Reasons.
3. 0x40 – 0x7F → Application-specific “non-fatal” NACK Reasons.
4. 0x80 – 0xBF → ONE-NET-specific “fatal” NACK Reasons.
5. 0xC0 – 0xFF → Application-specific “non-fatal” NACK Reasons.

Value	NACK Reason
0x00	**No Error**: Message is accepted as is or request is accepted / allowed. There are no problems. This is considered an ACK.
0x01	**Resources Unavailable**: Device cannot currently handle the request due to some lack of resources, such as RAM, or it is already servicing some other request.
0x02	**Internal Error:** A catch-all error for something that has gone wrong internally in the device.
0x03	**Busy Try Again:** The device cannot currently handle the request, but expects to be able to soon. The sending device is invited to re-submit the request in the near future.
0x04	**Busy Try Again Time:** Same as "Busy Try Again", but this message is accompanied by the time in milliseconds that the device expects to be busy.
0x05	**Bad Position Error:** Specified Position / Offset is invalid or not what was expected by the device.
0x06	**Bad Size Error**: Length/Size is invalid and/or does not mach what is expected by the device. Different from "Invalid Length Error", which means that the device either does not have the ability or refuses to HANDLE something the size requested.
0x07	**Bad Address Error**: DID or NID is invalid or undecodable.
0x08	**Invalid Max Hops**: Packet either has a "max hops" that is too large or there is some other problem dealing with max hops.
0x09	**Max Hops**: Packet either has a "hops" that is too large or there is some other problem dealing with hops.
0x0A	**Invalid Peer**: Some problem dealing with an invalid peer. Peer is not in the peer table, peer is out of range or cannot handle this message, etc.
0x0B	**Out Of Range**: Device is out of range, asleep, or cannot be reached
0x0C	**Route Error**: Message is not routed properly of the repeaters needed are unavailable
0x0D	**Invalid Data Rate**: Device is either using the wrong data rate, a data rate that is unavailable for whatever reason, or a data rate that not all devices in the transaction can achieve
0x0E	**No Response**: Message has not been responded to.
0x0F	**Invalid Message ID**: Message ID specified is invalid or does not match the one expected.
0x10	**Need Features**: Device needs the other device's features before it cannot proceed.

0x11	**Features Error:** General error concerning features / capabilities other than not having them. The devices' features / capabilities do not allow them to complete the transaction as planned.
0x12	**Bad CRC:** CRC in packet does not match the calculated CRC.
0x13	**Bad Key:** Sent when a device is using the wrong key.
0x14	**Already In Progress:** Similar to "Resources Unavailable", but more detailed. A transaction of the type requested is already in progress and the device either cannot handle another one or the device thinks that the request has already been approved and is already being processed and hence the request is unnecessary.
0x15	**Not Already In Progress:** Sent if / when a device attempts to proceed with a transaction that has not been set up yet (i.e. sending block or stream data before the block / stream transaction has been set up).
0x16	**Invalid Channel:** Channel used or requested is invalid for some reason. It is either a non-existent channel, a channel that cannot be used by the network, or a channel that cannot be accessed for whatever reason.
0x17	**Invalid Chunk Size:** An invalid chunk size has been specified in a block message. Either the chunk size is different from what was previously agreed to or the device wants to change chunk sizes.
0x18	**Invalid Chunk Pause:** An invalid chunk delay has been specified in a block message. Either the chunk delay is different from what was previously agreed to or the device wants to change chunk pauses.
0x19	**Invalid Byte Index:** An invalid byte Index has been specified in a block message.
0x1A	**Invalid Fragment Delay:** An invalid fragment delay has been specified in a block or stream message.
0x1B	**Invalid Priority:** An invalid priority has been specified in a message or the device wants to change priorities.
0x1C	**Permission Denied Non-Fatal:** Generic "Permission Denied" Error, but considered "non-fatal" so the device is allowed to try again if desired.
0x1D – 0x3D	**Reserved For Future Use**
0x3E	**Unset:** Reason is currently unspecified or unknown.
0x3F	**General Error:** General non-fatal NACK Reason where none of the other NACK Reasons fit.

Table 4.13 Non-Fatal NACK Reasons

Value	NACK Reason
0x80	**Invalid Length Error**: Device does not have the capability of handling a message or transaction with the length specified.
0x81	**Device Function Error**: Device cannot handle the request being given or the request does not make sense for this particular device. An example might be a message telling a device with no relays or switches to turn a relay or switch “off”.
0x82	**Unit Function Error:** A unit within a device cannot handle the request being given or the request does not make sense for this particular unit of the device. An example might be a message telling a unit that is not a switch or a relay to turn its switch “off”.
0x83	**Invalid Unit Error:** A message is addressed to a non-existent unit within a device.
0x84	**Mismatch Unit:** Unit exists and the message makes sense, but the device expected the message to be addressed to a different unit in the device.
0x85	**Bad Data Error:** Data portion of the payload is invalid or improperly formatted.
0x86	**Transaction Error**: Invalid transaction specified (such as a Block Data packet in the absence of a previous Block Request)
0x87	**Max Failed Attempts Reached**: Too many attempts and failures have occurred.
0x88	**Busy**: Device is busy. The sending device is not requested to try again in the near future, but instead should abort the message.
0x89	**No Response**: A timeout has occurred with no response.
0x8A	**Unit Is Input**: Similar to Unit Function Error, but more specific. A message addressed to an Input Unit has requested something that only an Output Unit can do.
0x8B	**Unit Is Output**: Similar to Unit Function Error, but more specific. A message addressed to an Output Unit has requested something that only an Input Unit can do.
0x8C	**Device Not In Network**: The packet refers to a device that is not in the network.
0x8D	**Device Is This Device**: A request has been made to pass something to another device that is actually the device that is being sent to. In other words, a catch-all NACK Reason for a message that can be returned to a sending device when the sending device sends a message to one device thinking it was a different device. This NACK Reason can also be used internally.
0x8E	**Sender And Destination Devices Are The Same**: A message has been sent where the source and destination devices are the same (a device is sending a message to itself).

0x8F	**Permission Denied Fatal**: Permission Denied. Do not attempt to repackage the request with different parameters. Subsequent requests will also be denied.
0x90	**Abort**: A transaction should be aborted.
0x91 – 0xBD	**Reserved For Future Use**
0xBE	**Unset:** Reason is currently unspecified or unknown.
0xBF	**General Error:** General Fatal NACK Reason where none of the other NACK Reasons fit.

Table 4.14 Fatal NACK Reasons

4.2.9.2.5 Acknowledgement Payload

Please see Table 4.12. The interpretation of the Acknowledgement Payload (if any) is determined by the Handle.

4.2.9.3 Route Packets

Please section 14 for more details on routing, route packets, and route payloads.

Pld. CRC 8 BITS	Message ID 12 BITS	Unused 4 BITS	Route Bits Payload 168 BITS	Encryption Method 2 BITS

Figure 4.15 – PCON Structure for Route Packets

4.2.9.4 Invite Packets

Please section 9.3 for more details on Invite packets. The PCON structure of Invite packets is as follows.

Pld. CRC 8 BITS	Version Number 8 BITS	Raw Device ID 12 BITS	Unused 4 BITS	Network XTEA Key 128 BITS	Master's Features 32 BITS	Encryption Method 2 BITS

Figure 4.16 – PCON Structure for Invite Packets

4.2.9.5 Block Data Packets

Please see the "Block And Stream Messages in ONE-NET" document for more details on Block Data packets. The PCON structure of Block Data packets is as follows.

Pld. CRC 8 BITS	Message ID 12 BITS	Chunk Index 6 BITS	Chunk Size 6 BITS	Byte Index 24 BITS	Data 200 BITS	Encryption Method 2 BITS

Figure 4.17 – PCON Structure for Block Data Packets

4.2.9.6 *Stream Data Packets*

Please see the "Block And Stream Messages in ONE-NET" document for more details on Block Data packets. The PCON structure of Stream Data packets is as follows. Stream Data packets are the only packet type in ONE-NET that does not use 32 rounds of encryption. Stream Data packets use only eight rounds of encryption due to the real-time nature of Stream Data transactions and the high resources required for encryption. Therefore Stream Data packets should be considered less secure than other packet types. ONE-NET users should take this into consideration and not send extremely sensitive information in Stream Data packets. Note that the "Response Needed" is interpreted as a Boolean value. Hence only one bit is needed. If the Response Needed bit is true, the sender will pause for an Acknowledgement from the recipient to make sure it is still receiving. Hence any device receiving this packet should respond if and only if the Response Needed bit is true.

Pld. CRC 8 BITS	Message ID 12 BITS	Unused 11 BITS	Response Needed 1 BIT	Elapsed Time 18 BITS	Data 200 BITS	Encryption Method 2 BITS

Figure 4.18 – PCON Structure for Stream Data Packets

4.3 Hops Field

The Hops field is present only in multi-hop packets. If present, it will be the last byte of the packet, immediately following the PCON. It is NOT encrypted and is NOT part of any CRC calculation. It consists of two 3-bit fields: Hops (the high three bits), and Maximum-Hops (the low 3 bits). Maximum-Hops is set by the sender of the packet and never changes. Hops is the number of hops taken so far in the packet's journey. It is initially set to 0. Every time a repeater repeats the message, the "Hops" field is incremented. A repeater will only repeat a message where "Hops" is less than "Maximum-Hops". by the sender, and decremented at each hop by the Client Repeater that retransmits it. A recipient of the packet can thus calculate the number of hops taken, by subtracting Remaining-Hops from Maximum-Hops. To improve efficiency, the number of hops needed is stored by each device so that the next time the two devices need to communicate, they will have a good guess of how many hops are needed.

Hops	Maximum-Hops
3 BITS	3 BITS

Figure 4.19 – Hops Field

5 Encoding, Cyclic Redundancy Checks, And Encryption

All ONE-NET packets use three methods to ensure packet integrity and security. Six-To-8-Bit Data Encoding and Cyclic Redundancy Checks help ensure that transmission errors are detected. Encryption helps ensure that only devices which are in the network can send and decipher messages.

5.1 Data Encoding

All transmitted data is encoded using a 6-bit to 8-bit mapping, in order to provide good clock synchronization and DC balance in the receiver (this also improves data accuracy, by introducing information redundancy). The data are encoded as a bit stream using the following mappings (where "Raw Data" are the raw 6-bit values to encode, and "Encoded Data" are the resulting 8-bit values).

Raw Data	Encoded Data
000000	10110100
000001	10111100
000010	10110011
000011	10111010
000100	10110101
000101	10111001
000110	10110110
000111	10110010
001000	11000100
001001	11001100
001010	11000011
001011	11001010
001100	11000101
001101	11001001
001110	11000110
001111	11000010
010000	00110100
010001	00111100
010010	00110011
010011	00111010
010100	00110101
010101	00111001
010110	00110110
010111	00110010

Raw Data	Encoded Data
100000	01010100
100001	01011100
100010	01010011
100011	01011010
100100	01010101
100101	01011001
100110	01010110
100111	01010010
101000	10010100
101001	10011100
101010	10010011
101011	10011010
101100	10010101
101101	10011001
101110	10010110
101111	10010010
110000	01100100
110001	01101100
110010	01100011
110011	01101010
110100	01100101
110101	01101001
110110	01100110
110111	01100010

011000	10100100
011001	10101100
011010	10100011
011011	10101010
011100	10100101
011101	10101001
011110	10100110
011111	10100010

111000	11010100
111001	11011100
111010	11010011
111011	11011010
111100	11010101
111101	11011001
111110	11010110
111111	11010010

Figure 5.1 – 6-bit to 8-bit Mapping

5.2 Encryption Method

All ONE-NET packets use XTEA encryption. XTEA encryption works in eight byte segments. Thus all ONE-NET packets have a payload section of the PCON that is a multiple of eight bytes. All packets except Stream Data packets use 32 rounds of XTEA encryption. Immediately after the encrypted portion of the PCON, there is a two bit UN-encrypted segment containing the encryption technique. Currently the only valid values of these two bits are 01, though other combinations can be used for debugging purposes and / or other experimentation purposes.

Bits	Single/Block Encryption Method
00	Reserved for debug purposes; unused in production Devices
01	XTEA-32
10	TBD
11	TBD

Table 5.1 – Single/Block Encryption

Note: In XTEA-XX encryption, the “XX” refers to the number of XTEA rounds used.

Due to the real-time nature of Stream Transactions, a different encryption method is used than for Single and Block Transactions. Stream Data Packets are sent using eight rounds of XTEA encryption. All other ONE-NET packets use eight rounds. Note that administration messages involving stream transactions as well as acknowledgments of Stream Data packets use 32 rounds of encryption, not eight.

Bits	Stream Encryption Method
00	Reserved for debug purposes; unused in production Devices
01	XTEA-8
10	TBD
11	TBD

Table 5.2 – Stream Data Encryption

5.3 ONE-NET CRC Calculations

Cyclic Redundancy Checks (CRC) are used to check packet integrity. ONE-NET uses an 8-bit CRC calculation scheme. The polynomial details are listed below in section 6.1. The files one_net_crc.h and one_net_crc.c are the C header and implementation files for CRC calculations.

ONE-NET uses CRC calculations in two places…

1. A CRC is calculated over the decrypted portion of the payload that is to be encrypted. The bytes over which the CRC is calculated are all but the first byte. The calculation is stored in the first byte. See section 6.2 for an example.
2. A CRC is calculated over all bytes of the packet starting with the Destination DID and ending with the encryption technique bits. Since the CRC is 8 bits and all bits need to be encoded, we throw away the two least significant bytes of the calculated CRC, then encode. See the example in section 6.2.

The polynomial used for the 8-bit CRC is 0xA6. It is computed over the raw PCON, from the Transaction Nonce field through the Data Payload field.

Order	8
Polynomial	0xA6
Reverse Data Byte	FALSE
Reverse Final CRC Before Final XOR	FALSE
Initial Value	0xFF
Final XOR Value	0x00
Check	0x6C (computed over the ASCII string “123456789” – 9 byte array 0x313233343536373839).

Table 5.3 – CRC Properties

5.4 ONE-NET CRC Example

A good way to understand CRC calculations is to follow an example. We will assume that device 003 (0xB4BA encoded) is sending device 004 (0xB4B5 encoded) a non-multihop, non-stay-awake single data packet of size 1. Let us assume that the decoded NID is 0x444555666. Encoded, that value is 0xC56A3CB53939. Let us assume that the seven byte raw payload, not including the CRC, is 0x22334455667788.

Recall that the PTYP (AKA PID) is calculated as follows…

The number of raw bits is 12. Let X denote a bit that has not been filled in yet.

XXXXXXXXXXXX

The left-most 4 bits are the size, which is 1.

0001XXXXXXXX

The Stay-Awake Bit is the next bit. Since this is not a Stay-Awake packets, that value is 0. The PID is now

00010XXXXXXX

The Multi_Hop Bit is the next bit. Since this is not a Multi-Hop packet, that value is 0. The PID is now

000100XXXXXX.

The remaining 6 rightmost bits are the packet type. In this case, this is a single data packet, so the type is 0. Filling in the rest of the bits, the 12 PTYP bits are

000100000000

Converting to hexadecimal, we have 0x101 as the Raw PTYP. Encoding 0x101, we get 0xB5BC as the Encoded PID..

Thus the packet layout is now as follows…

Preamble PATTERN 3 BYTES 0x555555	SOF PATTERN 1 BYTE 0x33	RPTR DID RAW – 12 BITS ENCODED – 0xB4BA	MSG CRC RAW – 6 BITS ENCODED – 8 BITS	DST DID RAW – 12 BITS ENCODED – 0xB4B5	NID RAW – 36 BITS ENCODED – 0XC56A3CB 53939	SRC DID RAW – 12 BITS ENCODED – 0XB4BA	PTYP RAW – 12 BITS ENCODED – 0xB5BC	PCON RAW – Varies ENCODED – Varies

Table 5.4 – CRC Calculation Step 1

The payload, please recall, is 0x22334455667788. The bytes to encrypt are therefore…

XX22334455667788

The XX represents the Payload CRC calculated over 0x22334455667788 using the polynomial constants in 5.1. The Payload CRC is calculated by calling the one_net_compute_crc in the one_net_crc.h and passing it the array(0x22334455667788), the array size(7), the initial value(0xFF), and the order(8). The result is 0x1E, which is the Payload CRC.

Now let's calculate the Payload CRC. We calculate this over the seven bytes of 0x22334455667788 and get the result of 0x1E. Filling in the X's, the payload to encrypt will be…

1E22334455667788

Let's assume the encryption key is 33333333-33333333-33333333-33333333. The number of encryption rounds is 32 since this is not a stream packet.

0x1E22334455667788 encrypts to 0x3F56E8F5142D7278. This is 64 bits. We must add the encryption method to the end, which is 0x40, which represents 32-rounds of XTEA-encryption. The relevant portion of the encryption technique are the leftmost two bits of 0x40, which is 01. We now have 66 bits to encode into 88 bits.

The 66 bits of 0x3F56E8F5142D727840 will be encoded into 88 bits (this appears to be 72 bits, but recall that we only care about the two most significant bits of 0x40 and are throwing away the other six bits, so we end up with 66 bits).

The result of encoding is 0xB253C46A3CB93956A9D45C.

Revisiting the packet, we have this so far.

Preamble PATTERN 3 BYTES 0x555555	SOF PATTERN 1 BYTE 0x33	RPTR DID RAW – 12 BITS ENCODED – 0xB4BA	MSG CRC RAW – 6 BITS ENCODED – 8 BITS	DST DID RAW – 12 BITS ENCODED – 0xB4B5	NID RAW – 36 BITS ENCODED – 0XC56A3CB 53939	SRC DID RAW – 12 BITS ENCODED – 0XB4BA	PTYP RAW – 12 BITS ENCODED – 0xB5BC	PCON RAW – Varies ENCODED – Varies

Table 5.5 – CRC Calculation Step 2

We are calculating the Message CRC, so we're only interested in what is to the right of it. Deleting some columns and filling min the PCON…

MSG CRC RAW – 6 BITS ENCODED – 8 BITS	DST DID RAW – 12 BITS ENCODED – 0xB4B5	NID RAW – 36 BITS ENCODED – 0XC56A3CB 53939	SRC DID RAW – 12 BITS ENCODED – 0XB4BA	PTYP RAW – 12 BITS ENCODED – 0xB5BC	PCON RAW – Varies ENCODED – 0xB253C46A3CB93956A9D45C

Table 5.6 – CRC Calculation Step 3

We now calculate the Message CRC over the bytes above using the same technique as last time…

The CRC of 0xB4B5C56A3CB53939B4BAB5BCB253C46A3CB93956A9D45C is 0xB0. Since we need to encode this, we can only use 6 of the 8 bits. We use the 6 most significant bits of 0xB0 the other two away.

0xB0 is 10110000 in bits. Keep 101100, which is 0x2C in hexadecimal. Encoding 0x2C, we get 0xCA. Filling this in, we get…

MSG CRC RAW – 6 BITS ENCODED – 0xCA	DST DID RAW – 12 BITS ENCODED – 0xB4B5	NID RAW – 36 BITS ENCODED – 0XC56A3CB 53939	SRC DID RAW – 12 BITS ENCODED – 0XB4BA	PTYP RAW – 12 BITS ENCODED – 0xB5BC	PCON RAW – Varies ENCODED – 0xB253C46A3CB93956A9D45C

Table 5.7 – CRC Calculation Step 4

5.4.1 CRC Example Final Result

The final result is therefore as follows.

0xCAB4B5C56A3CB53939B4BAB5BCB253C46A3CB93956A9D45C

Adding the Preamble, SOF, and Repeater DID back in, the packet to be sent is…

0x55555533B4BACAB4B5C56A3CB53939B4BAB5BCB253C46A3CB93956A9 D45C

The Message CRC byte is 0xCA, in bold blue above.

6 Sleeping Devices And “Stay Awake” Packets

ONE-NET is geared primarily towards battery-powered simple client sensor devices whose primary mode is sleep mode in order to preserve battery power. Generally these devices will wake up once an hour or once a day, send a quick status message, whereupon receipt of the message, they immediately go to sleep again. This poses problems when another device needs to send this device more information, such as an administration message like a key change, new settings, etc. The sleeping device will wake up, send its status message to the master, the master will ACK the status message, then the device will go back to sleep. The master then sends the device the new settings or the new key or whatever the new message is, but the message will never get there since the device has already gone to sleep. This is a problem.

One potential solution is to have the sleeping device pause before going to sleep just in case it needs to be contacted. This, however, wastes battery power if there is no further message, so is unacceptable.

The solution to this problem is one where the device does not need to stay awake if there are no more messages, but at the same time WILL stay awake if there are more messages. ONE-NET solves this problem by allowing “Stay-Awake” packets. Any device that needs to send either an Acknowledgement or a Data Packet and has more information than can fit into the packet being sent can set the “Stay Awake” flag (see Section 4.2.7, Figure 4.3) in the Packet Message Type field. Upon receiving a packet with the “Stay-Awake” flag set, a device will not go to sleep immediately, but instead stay awake with its transceiver on and listening. It will listen for packets and only go to sleep again after listening for a few seconds and not receiving any more packets. At that time, it will assume that all devices that need to contact it have already done so.

7 Keep-Alive Times, Flags, Keep-Alive Packets, and Key Verification

All devices are expected to "check in" with the master regularly. If the master does no hear from the client within the allotted time, the application-code is alerted. The interval that a device must communicate with the master at least once is called the "Keep-Alive" time. It is determined by the Master and is stored in Milliseconds. The Master will inform the Client of its Keep-Alive time as part of the Invite process. It can also change the Keep-Alive time at any point. Clients which have not communicated with the Master device within that interval are expected to send a "Keep Alive Response" message to the master. The "Keep-Alive Response" contains the last four bytes of the Network Encryption Key. If the key fragment sent is incorrect, the master will NACK the Client device and send it back the correct key fragment. The Keep-Alive Response message therefore serves three purposes…

1. It confirms that the Master and Client Devices are still able to communicate with each other.
2. It verifies that the Client is using the correct key.
3. It allows the Master to inform the master of any other transactions that need to occur.

What the Master does if the client does not check in on time will depend on the individual network. A common practice is to ping the client if it does not check in on time. If the client misses several check-in times, the Master may decide that it has broken or can no longer communicate and drop it from the network.

Sleeping devices must not have a sleep interval longer than its Keep-Alive time.

7.1 Flags (AKA Settings)

All devices are maintain a one byte "flags" component in memory. The flags bit should be considered as 7 bitwise Boolean values. Bit 7 is considered the high-order bit, bit 0 the low-order bit. Bits 1 through 4 are only relevant for devices which have Block or Stream capabilities. Bit 0 is unused. Below is a chart with the seven flags.

Bit	Name / Meaning / Interpretation
7	**ON_JOINED**: True if device is in a network, false otherwise
6	**ON_SEND_TO_MASTER**: True if the device should send status change updates to the master, false otherwise.
5	**ON_REJECT_INVALID_MSG_ID**: True if the device should reject messages that contain invalid Message IDs, false otherwise. See the "**Preventing_Replay_Attacks_And_Other_Security_Considerations_In_ONE-NET**" document for more details.
4	**ON_BS_ELEVATE_DATA_RATE**: True if the device should change elevate data rates for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.
3	**ON_BS_CHANGE_CHANNEL**: True if the device should change channels for "long" block and stream messages, false otherwise. See the "**Block And Stream Messages in ONE-NET**" document for more details.
2	**ON_BS_HIGH_PRIORITY**: If true, the device should use high priority queuing for "long" block and stream messages, If false, low priority should be used. See the "**Block And Stream Messages in ONE-NET**" document for more details.
1	**ON_BS_ALLOWED**: If false, "long" block and stream transfers are not allowed at all or this device. See the "**Block And Stream Messages in ONE-NET**" document for more details.
0	Unused

Table 7.1 – ONE-NET Flags Byte

8 Master Devices, Client Devices, Simple Clients, And Hybrid Devices

All ONE-NET networks have exactly one Master device. All other devices are client devices. All clients must check in with the master at regular intervals determined by the master. All key changes and flag changes for all clients goes through the master device. Adding clients to and removing clients from the network is always done through the master. The master assigns and approves all peer message setups, sends most other administration messages, and is in overall charge of the network. Therefore while not all devices need to be able to communicate with each other, all must be able to communicate with the master.

A device can have the ability to function as a Master, a Client, or either. If the device can function as both it is called a Hybrid advice. A Hybrid device, while in a network, will either always be the Master or it will always be a Client.

8.1 Simple Clients

A “Simple” Client is a Client device that does not have any of the advanced features. Define ONE_NET_CIMPLE_CLIENT if your device has very low RAM resources or code space. Simple Clients have the following restrictions.

1. They do not have any of the advanced timing options.
    a. They cannot “pause” messages or request a pause.
    b. Their Queue Level cannot be above MIN_SINGLE_QUEUE_LEVEL. Hence they cannot cancel messages based upon an expiration time or send messages in the future. All queued messages will be sent immediately.
2. They do not have many of the advanced capabilities.
    a. They cannot handle Multi-Hop, Route, Extended Single, Block, or Stream messages.
3. Acknowledgements
    a. As mentioned, they cannot send or receive requests to “slow down”, “speed up”, or pause.
    b. They are not informed of individual ACKs and NACKs. They are simply informed of the success or failure of the transaction as a whole.

    c. Hence they cannot change an ACK or a NACK. In particular, they cannot change a fatal NACK to a non-fatal NACK or vice-versa.

4. They cannot change channels or data rates after joining a network.
5. They can only handle Application messages of the ON_APP_MSG variety.
6. Security
    a. They cannot enhance security by making sure a client device does not "slide off" the client device table.
    b. They cannot request a key change.
7. They are not informed when a message is loaded / popped from the queue.
8. They are not informed of the recipient list of a message before sending, nor can they alter the recipient list.

The "Simple Client" bit is part of a device's Features. Other devices know not to attempt to communicate with a Simple Client if that communication involves any of the aspects on the list above. If the device needs to be able to do any of the above, do not define ONE_NT_SIMPLE_CLIENT.

9 Adding A Device To The Network

9.1 Information That Must Be Exchanged

To add a new client to the network, the following information must be exchanged in a secure way.

1. The Master must inform the joining device of the Network Encryption Key.
2. The Master must inform the joining device of the Master's Features.
3. The Master must inform the joining device of its Device ID.
4. The Master must inform the joining device of the device's Stay-Awake Interval.
5. The Master must inform the joining device of the device's flags / settings (see section 7.1).
6. The Master must inform the joining device of the device's fragment delays (relevant only if the joining device is capable of participating in Block or Stream messages).
7. The joining device must inform the Master of its features.
8. The master must inform the joining client that it has officially joined.
9. The Master must inform the joining device of the number of Multi-Hop Devices and Multi-Hop Repeaters in the network (relevant only if the joining device is capable of participating in Multi-Hop messages).
10. The Master must inform all other Multi-Hop-capable devices that a new device has joined, as well as inform them of the new number of Multi-Hop and Multi-Hop-Repeater capable devices in the network.

This information can be exchanged in one of at least two ways: ONE-NET wireless messaging or some application-specific method. Examples of application-specific methods might involve either physically connecting the two devices and passing the information through a NON-wireless connection or by physically walking between the two devices with a thumb drive or something similar and transferring the information that way.

9.2 Invite Keys

Assuming the invitation process is done via ONE-NET wireless messaging, the client device being added must have an encryption key other than the Network Encryption Key known both to the Master and the Device being added as a Client. This called the client device's "Invite Key" and it will be used by the Master to encrypt the Invite New Client Packet. Note that the Network Encryption Key is contained in the Invite New Client Packet. Hence it is vital for security purposes that no one but the Master device and the device being added knows what the Invite Key is. How the Master learns what the Invite Key is is application-specific and beyond the scope of this document, but in general the Master will learn the Client's Invite Key from a person typing it in via the Command Line Interface.

The Invite Key is 128-bit XTEA encryption key, just as the Network Encryption Key is. The Master will encrypt the Invite Packet using this one-time key. The client being added will use this key to decrypt. The only time this key is ever used is when encrypting and decrypting Invite Packets. As with all other non-Stream packets, Invite Packets use 32 rounds of XTEA encryption. Invite Packets are the only ONE-NET packets that do not contain Message IDs.

Generally, Invite Keys are shown as ASCII strings. Also, generally, the first 64 bits of the Invite Key will be the same as the last 64 bits. For ease of reading, zeroes and ones are disallowed, as are all characters that can be misinterpreted as zeroes and ones to prevent the digit 0 and the letter O or the digit 1 and the letter L or the letter I. Hence valid ASCII values are upper or lower case A – Z (excluding upper and lower case I, L, and O), and 2 – 9).

Hence 2345-678A would be a valid Invite key. To convert to an hexadecimal XTEA key, the characters should be copied so there are 16 of them…

2345-678A-2345-678A and each should be converted from ASCII to hexadecimal. The corresponding hexadecimal key would be the following…

32333435-36373841-32333435-36373841

because '2' is 0x32 in ASCII, '3' is 0x33 in ASCII, etc.

9.3 Invite Packets

The Invite Packet encrypted payload contains 3 XTEA blocks of 64 bits, so there are 192 encrypted bits. As with all packets, the payload CRC makes up the first 8 bits, so there are 184 bits remaining. Only 180 bits of the 184 remaining encrypted bits are used. The Encryption Method, as always, is not encrypted.The makeup of the PCON of an Invite Packet is as follows. The version as of ONE-NET 2.3.0 is Version 2.

Pld. CRC 8 BITS	Version Number 8 BITS	Raw Device ID 12 BITS	Unused 4 BITS	Network XTEA Key 128 BITS	Master's Features 32 BITS	Encryption Method 2 BITS

Figure 9.1 – PCON Structure for Invite Packets

Looking at the list of 10 things that must happen during the Invite Process in Section 9.1 above, note that list items 1, 2, and 3 are contained in the Invite Packet. Thus upon decrypting the packet above, the Client will know its Device ID, the Network Encryption Key, and the Master's Features.

A device looking to join the network will either scan channels (one per second) listening for Invite Packets, or if it knows the channel to listen for (usually via Command Line Interface), will change to the Network Channel. Once it receives an Invite Packet that decrypts properly, it will know the Network Channel and copy it to its memory, along with its new Device ID, the Network Encryption Key, and the Master's Features.

To initiate the process, the Master sends an Invite New Client Packet when a new Client joins the network. It is important to note that this message is sent to a specific Client; the Master does **not** query for Devices to add. When a Device is being added to the network, the Master sends an Invite New Client Packet at least three times per second. The Master will continue to do this until the Client has responded with a "Keep Alive Response" or a specified time out period has expired. If Multi-Hop-Repeaters are available, every fifth Invite packet sent by the Master will be a Multi-Hop packet. In this way, devices which are not within range of the Master can be added.

When ready to join, the Client joins the network by scanning channels until it hears the Master Invite New Client Packet. The Client listens for one second on each channel. When it hears the Master Invite New Client Packet, the Client sends a Status Response admin packet to alert the Master that it has joined the network, and report its capabilities. When it receives a valid Invite Packet, it will respond with a "Keep Alive" packet.

9.4 Client Response to Master's Invite Packet

Once the client receives the Invite Packet, the Invite Key is discarded. All further packets are encrypted and decrypted using the Network Encryption Key that was contained in the Invite Packet.

The Client will respond to the Invite Packet with a Keep-Alive Response packet. Since the Master does not have the Client's features, this packet will be NACK'd by the Master. The NACK Reason will explain that the Master needs the Client's features. The Client will send its features, then immediately upon getting an ACK, will re-send the Keep-Alive Response. The Master will likely NACK that packet and give the Client device a new Message ID. The client will repackage the Keep-Alive Response with the proper Message ID. The Master will respond to each Keep-Alive Response message from the client with an administration message in the payload containing either the fragment delays (if the Client is Multi-Hop Capable), the flags, the Keep-Alive time, and finally, an ON_ADD_DEV message saying that the client has officially been added. This message will contain the final bit of information that the Client needs (the number of multi-hop and multi-hop repeater devices – needed only if the Client is Multi-Hop capable). At this point, the Client and Master have exchanged all the information they need and the Client has been added. The Client will send a final Keep-Alive message, which the Master will respond to with a simple ACK. This is the signal to the Client that no more messages need to be sent.

The master will now attempt to inform all non-sleeping devices in the network that a new device has been added. This message will contain the new device's Device ID and the total number of multi-hop devices and multi-hop repeaters in the network. There is a five second pause between the notification of each device in case that device needs to contact the new device.

10 Deleting A Device From The Network

Deleting a device is simpler. The Master sends an ON_RM_DEV message to the device to be deleted, the device ACKs, and the Master removes that device from its table, including its peer table. The device that was removed now looks for a new network to join. As with adding a device, the Master will inform all non-sleeping devices that the device has been removed. This message payload contains the deleted device's Device ID as well as the number of multi-hop-capable and multi-hop-repeater capable devices. All Clients should remove this device from their peer and device tables.

11 Data Rates And Channels

All devices must have the ability to communicate at the 38,400 data rate. Some devices may choose, or the master might tell them, to change data rates and / or channels. Generally this is done to accommodate long block or stream message transfers. The main reason for this is to prevent the block or stream transactions from completely dominating the airwaves and hindering message traffic between other devices. In addition, devices can more easily ignore traffic that does not concern them. Channel and / or data rate changes can also take place independently of any block / stream transactions.

Channel / Data Rate changes have both pros and cons. A case by case weighing of the pros and cons is needed to decide whether and when to change data rates ad channels.

Changing to a higher data rate is useful in order to transfer data much more rapidly than is possible at the slower base data rate. Again, this is generally done in block and stream transactions.

The pros and cons are generally the same. If two devices are not functioning at the same data rate and channel, then cannot hear each other. The possible "pro" in this was mentioned above: no transaction will prevent other transactions from getting through. The "con" is obvious. If two devices NEED to communicate with each other and are at different data rates / channels, one of them must change data rates and channels or the transaction must wait for the device that changed data rates and channels to change back. This increases both complexity and latency. It increases complexity because the master or some other device must maintain two more pieces of information in its device information table (channels and data rates of other devices). It also possibly involves additional query messages to find out data rates and channels, then the data rate and channel switch itself, which adds to the latency. Hence a case-by-case decision needs to be made by the master and possibly other devices in the network. What are the consequences of a congested bandwidth? What are the costs of increased latency of short, single messages. When it is absolutely critical that all devices must be able to communicate quickly with all other devices, data rate and channel changes should not be allowed. An additional consideration to consider is the range and reliability when broadcasting at different data rates, as well as packet lengths. Longer packet lengths and higher data rates will reduce the physical range of the messages as well as the reliability. Thus one cannot assume that two devices that can reliably communicate without Multi-Hop Repeaters when transmitting short Single packets at the low base data rate will be able to do so at higher data rates or when transmitting longer packets.

All devices, within their Features bytes, contain the following information.

1. Whether they have the ability to change data rates or channels
2. What data rates they can operate at.

Most networks will have no need for devices switch data rates or channels.

Section 12 contains more information on data rates and other transceiver issues.

12 RF Specification

12.1 USA Frequencies

Band limits: 902-928 MHz Frequency is defined as the center of the modulation bandwidth. The large FSK deviation and digital modulation allows the ONE-NET physical layer to operate under Digital Modulation Spread Spectrum (DTS mode) characteristics of the FCC regulatory specifications 15-247.

Channel	Frequency (MHz)
1	903.0
2	904.0
3	905.0
4	906.0
5	907.0
6	908.0
7	909.0
8	910.0
9	911.0
10	912.0
11	913.0
12	914.0
13	915.0
14	916.0
15	917.0
16	918.0
17	919.0
18	920.0
19	921.0
20	922.0
21	923.0
22	924.0
23	925.0
24	926.0
25	927.0

Table 12.1 USA RF Frequencies

12.2 European Frequencies

Band limits: 865–868 MHz Frequency is defined as the center of the modulation bandwidth. The large FSK deviation allows the modulation to fall under Digital Modulation Spread Spectrum characteristics of the ETSI regulatory specifications.

Channel	Frequency (MHz)
1	865.8
2	866.5
3	867.2

Table 12.2 – European RF Frequencies

Modulation is two-level FSK. A data one is defined as the upper deviation frequency (positive frequency deviation) and a data zero is defined as the lower deviation frequency (negative frequency deviation).

12.3 Deviation

Single sided deviation is ±240 kHz.

Total deviation is 480 kHz.

Tolerance of total deviation is ±35 kHz.

Tolerance of frequency center for the total deviation is ±17.5 kHz.

12.4 List Of Data Rates

Speed	Rate (kbps)	Inclusion
Base Speed	38.4	mandatory
High Speed 1	76.8	optional
High Speed 2	115.2	optional
High Speed 3	153.6	optional
High Speed 4	192.0	optional
High Speed 5	230.4	optional

Table 12.3 – Data Rates

The data rate must be accurate to ±1.5%.

12.5 Transmit Output Power

Transmit output power is defined as transmitter antenna gain plus input power to antenna.

Transmitter antenna gain is measured in decibels relative to an isotropic antenna (dBi).

Transmit output power is measured in decibels referenced to one milliwatt (1 mW).

Master Devices must have a minimum output power of 12 dBm. An example would be input power to antenna of 10 dBm plus antenna gain of 2 dBi, yielding effective radiated transmit output power of 12 dBm.

Client Devices must have a minimum output power of 8 dBm. And example would be input power to antenna of 10 dBm plus antenna gain of -2 dBi, yielding effective radiated transmit output power of 8 dBm.

12.6 Receiver Sensitivity

Receiver sensitivity is defined as the rated sensitivity at receiver matching network plus receiver antenna gain.

Receiver antenna gain is measured in decibels relative to an isotropic antenna (dBi).

Receiver input power is measured in decibels referenced to one milliwatt (1 mW).

Master Devices must have a minimum receiver sensitivity of -102 dBm. And example would be receiver sensitivity of -100 dBm plus antenna gain of 2 dBi, yielding effective receiver sensitivity of -102 dBm.

Client Devices must have a minimum receiver sensitivity of -99 dBm. And example would be receiver sensitivity of -101 dBm plus antenna gain of -2 dBi, yielding effective receiver sensitivity of -99 dBm.

13 Message Priorities

Message priorities in ONE-NET are not intentionally not defined precisely so that network designers can decide what constitutes a high- versus low-priority message. ONE-NET uses a priority queue for outgoing messages, so a high-priority message in the queue will be sent before a low priority message even if the low priority message was pushed onto the queue first. In addition, low-priority Block messages will allow high-priority Single messages to pause the Block message.

14 Routes

Route messages can be thought of as “ping” messages. One device pings another. From the ping response (or lack of a ping response), it determines whether it can communicate with the other device, how long it is expected to take, how many hops are in between, and what repeaters are needed.

14.1 Route Payloads

A Route Payload is a 21 byte / 168 bit payload containing the raw Device IDs involved. Since each Raw Device ID is 12 bits, 14 raw Device IDs can fit in a Route payload. A Route Payload is a NULL-terminated list of Raw IDs in the route. “000”, the raw Broadcast Device ID, signifies the end of the message. When repeating or responding to a Route Message, each Device appends its Raw DID to the payload. If its Raw Device ID is already present in a “leg” of the route, the message is ignored.

A “route” will have two “legs”: the “to” leg and the “return” leg. All route packets will have a known destination. If that destination is already in the route payload, the packet is in its “return” leg. If not, it is in its “to” leg.

A trivial example of a route is them non-multi-hop example, Assume that 005 wants to send a route message to 008.

1. The payload is 21 bytes / 168 bits. The route payload is filled with all 0’s.
    a. 000-000-000-000-000-000-000-000-000-000-000-000-000-000
2. As the initiator, 005 starts the message off by filling in its raw DID, then it sends the message.
    a. 005-000-000-000-000-000-000-000-000-000-000-000-000-000
3. Device 008 receives the message. Recall that route payloads are NULL-terminated. Hence the message 005-000-000-000-000-000-000-000-000-000-000-000-000-000 should be interpreted as an array of Raw DID’s. In this case, since there is only one raw DID before the first zero DID, the array size is 1, and contains the source (005). It does NOT contain 008, so 008 appends its DID and passes it along…
    a. 005-008-000-000-000-000-000-000-000-000-000-000-000-000

4. Device 005 (the original sender) receives the message and notes that it was the original sender and that the destination is in the message, so the route message is complete and was successful. It may have kept track of the time that it SENT the route message, so simply taking the current time and subtracting the original time provides the length of time the trip took. It takes the route packet and throws out its DID(005) and the Recipient's DID(008). The DIDs that remain are the repeater DIDs. I this case there are none.

14.2 An Example Involving Repeaters

This time we'll assume that Device 005 is sending to 008 and 008 is not within range. Let's assume that 006 and 007 are client repeaters, 006 is within range of 005 and 007 and 007 is within range of 006 and 008. (Note: always keep in mind that the route message is an array (or perhaps even better, a linked list) of 12-bit Raw Device IDs, where 000 represents "End Of Array" / "End Of List".

1. 005 sends a route message from 005 to 008. The route message is 005-000-000-000-000-000-000-000-000-000-000-000-000-000
2. 006 receives the message and checks whether it is the source or the recipient and whether it is already on the list. It is not the source or recipient and not on the list, so it appends its DID and forwards the message. 005-006-000-000-000-000-000-000-000-000-000-000-000-000
3. 007 receives the forwarded message and checks whether it is the source or the recipient and whether it is already on the list. It is not the source or recipient and not on the list, so it appends its DID and forwards the message. 005-006-007-000-000-000-000-000-000-000-000-000-000-000
4. 008 receives the forwarded message and checks whether it is the source or the recipient and whether it is already on the list. It is the recipient and it is not on the list, so it appends its DID and replies to the message. 005-006-007-008-000-000-000-000-000-000-000-000-000-000
5. 007 receives the reply message and checks whether it is the source or the recipient and whether it is already on the list. It is not the source or recipient. It IS on the list. It is on the list exactly once. It searches the list for the destination (008) and finds it. It now checks to see whether it (007) is in the return "leg" of the message. In other words, it checks whether it is in the list AFTER 008. It is not, so it should append itself to the RETURN

leg (it’s already on the “to” leg) and forward the message.
005-006-007-008-007-000-000-000-000-000-000-000-000-000

6. 006 receives the reply message and checks whether it is the source or the recipient and whether it is already on the list. It is not the source or recipient. It IS on the list. It is on the list exactly once. It searches the list for the destination (008) and finds it. It now checks to see whether it (006) is in the return “leg” of the message. In other words, it checks whether it is in the list AFTER 008. It is not, so it should append itself to the RETURN leg (it’s already on the “to” leg) and forward the message.
005-006-007-008-007-006-000-000-000-000-000-000-000-000
7. 005 receives the reply message and checks whether it is the source or the recipient and whether it is already on the list. It is the source. It now looks at the list to see whether the destination(008) is on the route. It is. The route is therefore successful and complete. Implicitly, route 005 will append its DID (since the route message process is complete, it probably won’t bother). The route is
005-006-007-008-007-006-005-000-000-000-000-000-000-000.
8. Removing the 0 entries, the route is 005-006-007-008-007-006-005.
9. Splitting this into the two legs, we get…
    a. 005-006-007-008
    b. 008-007-006-005

From these two legs, device 005 knows that if it wants to communicate with device 005, it should expect the message to take two hops and it should expect to need 006 and 007 to function as repeaters. Please note the comments involving data rates and packet lengths discussed in Section 10. Routes may not always be constant due to data rates, packet lengths, time of day, environmental factors, network density, and devices moving around. This will vary considerably from network to network.

15 Message Status Codes

The following message codes are used both internally and externally by ONE-NET. The name of this enumerated type is on_message_status_t. Many of these values are fairly loosely defined, but can give extra meaning to information. Often the meaning will be apparent without needing these codes if there is an accompanying NACK Reason, which can be more descriptive. Regardless, these codes are defined and can be used to add more information. They are primarily used only internally by ONE-NET.

Value	Message Status Code
0x00	**Default Behavior**: The "Default Behavior" should be used. "Default Behavior" is intentionally undefined as the meaning will vary greatly with conditions.
0x01	**Message Continue**: The message should be "continued". Again, this is intentionally largely undefined, but in general it means that the transaction or message is incomplete and should continue and not be aborted.
0x02	**Message Abort:** The attempt is incomplete and should be aborted. No "failure" or "success" is attached with this code.
0x03	**Message Success:** The message is complete and has succeeded.
0x04	**Message Fail:** The message is complete and has failed.
0x05	**Message Respond:** A message has been received and should be responded to. Primarily an internal ONE-NET code.
0x06	**Message Timeout**: Message is complete and has failed for the reason of a timeout.
0x07	**Message Ignore:** A message has been received and should be NOT be responded to. Primarily an internal ONE-NET code.
0x08	**Message Accept Packet**: Packet has been received and is accepted.
0x09	**Message Reject Packet**: Packet has been received and is rejected.
0x0A	**Message Accept Chunk**: Chunk of a Block Transfer has been received and is accepted.
0x0B	**Message Reject Chunk**: Chunk of a Block Transfer has been received and is rejected.

0x0C	**Message Terminate**: Message / transfer / transaction / process should terminate. No “failure” or “success” is attached to this code.
0x0D	**Message Internal Error**: Some Internal Error has occurred while processing a message.
0x0E	**Message Route Unavailable**: Packets cannot reliably be sent from one device to another at this time.
0x0F	**Message Block/Stream Setup Change**: The parameters of a Block/Stream transfer either have been or need to be changed and a new setup message must be sent.
0x10	**Message Pause**: Message either is paused or needs to be paused
0x11-0x7D	**Reserved For Future Use**
0x7E	**Message Status Unset:** Catch-all message code where none of the others fit.
0x7F	**Message General Error:** Catch-all general error message code where none of the others fit.
0x80-0xFF	**Application Specific:** These codes are reserved for application-specific codes

Table 15.1 ONE-NET Message Codes

16 Application-Level Code

Developers implementing ONE-NET must write the following application-level code. See the one_net_port_specific.h, one_net_client_port_specific.h, and one_net_master_port_specific.h files for more details. See the Evaluation Board project code for examples. Note that not all of these functions will need to be written and many implementations will be empty.

1. one_net_adjust_recipient_list: Called when a message is about to be sent. Informs the application level code of the addressee(s) of the message and allows the application code to change the addressee(s) or cancel the message. Not needed only for “simple” devices.
2. one_net_single_msg_loaded: Called every time a message is about to be sent. Allows the application code to change or abort the message. Not needed only for “simple” devices.
3. one_net_adjust_fatal_nack: Allows the application code to overrule ONE-NET’s decision about whether a NACK is “fatal” or not. Not needed only for “simple” devices.
4. one_net_master_handle_single_pkt: Called every time a single packet is received. Needed only for Master devices.
5. one_net_client_handle_single_pkt: Called every time a single packet is received. Needed only for Client devices.
6. one_net_master_handle_ack_nack_response: Called every time an ACK or NACK is received for a Single Data packet. Needed only for master devices.
7. one_net_client_handle_ack_nack_response: Called every time an ACK or NACK is received for a Single Data packet. Needed only for non-“simple” client devices.
8. .one_net_client_txn_status: Called when a Single message tranaction is complete. Needed only for client devices.
9. one_net_master_ txn_status: Called when a Single message tranaction is complete. Needed only for master devices.
10. one_net_master_update_result: Called when a master has finished sending an administration packet to a client. Needed only for master devices.

11. one_net_master_invite_result: Callback for an Invite attempt. Reports the result to the application code.
12. one_net_master_client_missed_check_in: Called when a client device has not reported back within the specified time period. Needed only for master devices.
13. one_net_master_device_is_awake: Called when a client device has checked in with the Master. Needed only for master devices.
14. one_net_client_invite_result: Called when this Client has joined a network. Client devices only.
15. one_net_client_client_removed: Called when any client has been removed from the network. Client devices only.
16. one_net_net_client_client_added: Called when any client has been added to the network. Client devices only.
17. one_net_client_reset_client: Called when a client is NOT a member of a network yet or has just been removed. Client devices only.
18. one_net_client_get_invite_key: Called to retrieve the invite key that the Client should use to decrypt invite messages. Client devices only.