1 Table of contents

2 Introduction

The **Optin Sensor Protocol** is a payload-agnostic connection-oriented Client-Server protocol with small overhead used in IoT applications.

- Its simplicity makes it a perfect choice for devices with scarcer resources.
- It provides numerous optional solutions to guarantee that the critical packets reach their destination and that the number of lost packets is reduced, even in non-reliable networks.
- The handling of non-critical packets is not burdened by the restrictions that come with the application of the security and reliability tasks.
- The optional security features of the protocol allow the usage of complete packet encryption, ensure message integrity, and message-wise authentication of both Client and Server.

Attention: the device-specific content of the *DATA* messages can be found in the appendix.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

The target audience for this document is primarily individuals who implement OSP or who architect systems that will use this technology. This document assumes that the reader is familiar with the Internet Protocol (IP), related networking technology, and general information on system security terms and concepts.

3 Legal Notice

The Optin Kft. (henceforth Author) provides access to use or implement the OSP, to copy or publish the protocol-specification under the conditions that EVERY copy of the protocol-specification or any part that shows up in other documents, specifications, articles or every system that implements the OSP contain the followings:

- The name of the Author:
  (Optin Kft.)
- a link or URL to the OSP specification on the Author's website:
  www.en.optin.hu/products-services/optin-sensor-protocol/

4 System Overview

The smallest unit of data the protocol handles is an OSP packet. The packets can be embedded into other protocols and other protocols can be embedded into specific packets as payload (See: DATA and COMMAND packets). This specification doesn't define whether these packets should be sent as a whole or divided into smaller chunks.

OSP is a connection-oriented protocol, meaning that a communication session MUST be established (See: clause 6) between the communicating parties before data can be exchanged. Packets received while no OSP session is active MUST be discarded. The life of an OSP session does not depend on, and is not linked to the transport layer protocol used to handle the communication.

OSP is also a Client-Server protocol, meaning that a session can only be established between exactly two parties: the Client and the Server. A session MUST be started by the Client, but MAY be closed by any parties. Both parties have a specific role during connection establishment, management, and data transmission.

4.1 Packet structure

Every packet consists of three parts: a fixed header, the OPTIONAL body of the packet, and an OPTIONAL message authentication code, or tag at the end of the packet. The fixed header identifies the session, contains a sequence number, identifies the type of the packet, includes flags that define the state and the total size of the packet. The packet body MAY contain a specialized header and data payload. The MAC is an OPTIONAL cryptographic message integrity and authentication code, that is present only if security features are enabled.

Fixed header	SID
	SequenceNum
	MsgType + Flags
	PacketSize
Packet body	SpecHeader
	Data payload
MAC (opt)	Message authentication code

4.2 Definitions

4.2.1 In-flight uniqueness

A value y with semantics X is "in-flight-unique", if there is no packet at a given moment and in a given direction on the communication channel, nor in the communication caches or memories that's X value is also y.

Examples are the MessageID and the CommandID. The in-flight-unique values help to identify the packets unequivocally in a window of time defined by the receiver.

4.2.2 Identifiers

The identifiers are non-negative numbers. The value 0 is always reserved and indicates a special condition - in most cases an error or an invalid entry.

4.2.3 String/binary types

String or binary data can only occur at the end of the packets, preceding the OPTIONAL, fix-sized MAC, its length is unlimited and there is no closing byte. The end of the string can be calculated using the packet's PacketSize field.

In terms of the protocol the strings and binaries are treated equal: byte-sequences with arbitrary length.

4.2.4 Client and Server

The Client is the communicating party that MUST initiate the communication session and connect to the Server. It is also the Client only that can pause an active OSP session.

The current 2.0 version of the protocol restricts the types of packets that communicating parties can assemble and send. DATA packets can only be composed by the Client (like the measurement results of a sensor or set of sensors) and sent to the Server, which acts as a data collector. RESEND requests and ACKNOWLEDGE packets can only be sent from the Server to the Client.

The COMMAND and FIRMWARE features, intended to be used for controlling, configuring, or updating the firmware of a Client device are based on request-response mechanisms. Commands can only be initiated by the Server, while firmware requests can only be issued by the Client.

4.2.5 Byte order

In the OSP, like in most network protocols, big-endian format is used for numeric values with more than one bytes. This means that the most significant byte (MSB) is sent first and the LSB is sent last.

The only exception from this rule is the PacketSize field, that has a special encoding.

4.3 Security

There are two operating modes of the protocol, depending on security needs: non-secure and secure.

The protocol does not provide the means for the Client to communicate its security settings during connection establishment, as all settings and parameters should be available to the Server linked to the DeviceType and ModuleID of each possible Client device.

There is also no protocol-level method for secure key-distribution, as it is assumed, that all security settings are set both on the Server, and the Client devices before device deployment. If security is compromised, it is highly RECOMMENDED to update the security settings (like encryption key) using a physical/non-OSP dependent method.

Of course, it is highly RECOMMENDED to change the security settings at a regular interval, even if no security breach is suspected - COMMAND packets MAY be used for this (see clause 7.).

The minimum set of information that MUST be available to the Server about each device includes the following:

- ModuleID
- DeviceType
- OSP version used
- Security features enabled/disabled

- If security is enabled:
    - Block cypher algorithm used
    - Block/Key size
    - Encryption key
    - MAC (Message Authentication Code) size

A specific Client device MUST be uniquely identifiable using the combination of its DeviceType and ModuleID.

4.3.1 Non-secure communication

If the security of the communication is not a concern, the protocol can be employed without any security features enabled. In this case, session establishment is much simpler, all packets are forwarded without encryption, and the MAC field at the end of each packet is omitted.

4.3.2 Secure communication

If a secure connection is needed, both session establishment, and later message encryption and authentication uses symmetric cryptographic techniques. It is RECOMMENDED to use the AES encryption algorithm with 128bit block/key size.

The session establishment process uses a four-way handshake based on an ISO/IEC 9798-2 method (Menezes et.al. Handbook of Applied Cryptography p. 402) to provide protection against most attack types and provide mutual authentication of the Client and the Server (see clause 5).

During an active session the EAX (link) method MUST be used to protect most packets. This method ensures payload confidentiality, integrity, and per-message authentication of communicating parties.

In each message where EAX is enabled (E == 1), the Fixed Header MUST be forwarded as plaintext (Associated Data), while the payload of the message MUST be encrypted using the pre-distributed symmetric encryption algorithm parameters, block size, and key in EAX cipher operation mode. As per the EAX method, a MAC MUST be generated and appended to the end of each message to provide authenticity and integrity checking for the Associated Data and the payload as well. It is RECOMMENDED to use a MAC code of at least 64 bits.

The nonce used during encryption and MAC generation of the EAX method is quasi-unique to each message (preferably a nonce is never used more than once with the same encryption key), and MUST be equal to a combination of the initial vectors shared during connection establishment, increased by the SequenceNumber for each message in a given communication direction (ClientInitVector+ ServerInitVector for the Client » Server direction, and ServerInitVector+ ClientInitVector for the Server » Client direction, where '+' means concatenation. See clause 5.1).

If a communicating party receives an EAX-enabled packet, as per the EAX method, the MAC MUST be checked before decrypting the message payload. If the authentication fails, the packet MUST be ignored. If authentication succeeds, the payload SHOULD be decrypted, and the packet can be further processed.

5 Fixed header

Every packet MUST start with a fixed header, which is a minimum of 6 bytes and a maximum of 7 bytes long depending on the size of the field called PacketSize. The format of the fix header:

Index	4-7 bit	3 bit	2 bit	1 bit	0 bit
0 byte	SID (2 bytes)				
2 byte	SeqNum (2 bytes)				
4 byte	MsgType	Cached	Saved	AckReq	EAX
5 byte	PacketSize ([1-2] bytes, chaining)				

5.1 Session Identifier (SID) (2 bytes)

The SID identifies an OSP communication session. The value of the session identifier is defined by the Server in the first CONNECT response (the SID of the initial Client » Server CONNECT packet MUST always be 0). After the first CONNECT response, all packages in both the uplink and downlink directions MUST contain the same SID. The SID generated by the Server SHOULD be random and MUST be unique (in a sense that no two active sessions can be running with the same SID at any time).

One Client device can only manage a single OSP session at a time. If a Client that has an associated SID closes its session its SID MAY be freed up and used to identify new sessions. Also, if a device that has an associated SID succeeds in opening a new session, its previous SID MAY be freed up. The SID MAY be used to stretch an OSP session over longer timeframes, independent of lower layer protocols (like TCP/UDP).

5.2 SequenceNumber (SeqNum) (2 bytes)

A two-byte unsigned value that MUST be unique to all messages sent in a direction during a session, regardless of message semantics. Its main purpose is to make it harder to execute replay attacks against the protocol and protect the session in cases of message duplication.

It MAY also be used both on Server and Client side to create session statistics, or to detect the loss or ordering errors of packages.

The initial value of the sequence number MUST be 1. From the very first packet of each session, the sequence number of the uplink and downlink directions are treated separately.

The sequence number MUST be incremented by 1 every time a packet is sent by the party, regardless of packet semantics. The sequence number MUST NOT overflow, before this happens, the connection MUST be closed (and a new connection MAY be established).

It is RECOMMENDED to use a window size (W) of 32. If a party receives a packet with sequence number N, only packets with sequence number M SHOULD be accepted afterwards, where:

- M > N (and N becomes M), or
- (M > (N-W)) && (M < N), and no packet with sequence number M has been accepted before (during the current session).

5.3 Message Type (MsgType) (4 bits)

A 4 bit unsigned value, marking the type of the packet, that MUST match one of the following values:

MsgType's value	Short name	Description
0x00	[RESERVED]	Not in use!
0x01	CONNECT / DISCONNECT	A packet initiating a connection or disconnection with the Server
0x02	COMMAND	Command type packet
0x03	ACKNOWLEDGE	Acknowledgement packet for an earlier packet to be acknowledged
0x04	PINGREQ	Ping request
0x05	PINGRESP	Ping response
0x06	FIRMWARE	Firmware packet
0x07	RESEND	Requests resend
0x08	DATA	General data packet
[0x09 – 0x0F]	[RESERVED]	For further development

5.4 Cached bit (C) (1 bit)

This bit MAY only be set in DATA type packages, it MUST be 0 in all other message types. If this bit has a value of 1, then it's a packet that was attempted to be sent by the device at least once. This is possible in the following situations:

- If the Server requested a packet to be resent with a RESEND message.
- If the receipt to the packet to be acknowledged has not yet arrived and it was resent automatically.
- The Client detected that the packet was not sent and tries to send it again.

5.5 Saved bit (S) (1 bit)

This bit MAY only be set in DATA type messages, it MUST be 0 in all other types of messages. If this bit is set to 1 it means the packet was archived earlier, and contains data generated at an earlier time.

Depending on the application, configuring this bit may be justified when sending archived data packets after the following situations:

- No valid session could be established for an extended time, and the device could not forward the data packets to the Server.
- While roaming, data forwarding was disabled or limited and the device was forced to archive all or part of the generated data.
- Data forwarding was disabled then enabled again.

5.6 AckReq bit (A) (1 bit)

This bit MAY only be set in DATA type messages, it MUST be 0 in all other types of messages. If this bit is set to 1, it means the reception of the packet must be acknowledged by the Server by sending an ACKNOWLEDGE packet to the Client.

5.7 EAX enabled (E) (1 bit)

If this bit is set, it means the packet body is encrypted, and the packet ends with a MAC code, all done using the EAX method (see clause 3.3.2).

If secure mode is disabled, this flag MUST NOT be set for any packets. If secure mode is enabled, this flag SHOULD be set for most packets.

5.8 PacketSize (1 or 2 bytes)

The size of the complete message in bytes, including the fixed header, the payload, and the optional MAC section. This field's size is between 1-2 bytes. The first byte's highest-order bit (8. bit) is not a part of the represented number. If this is 1 the following byte also becomes a part of the length field. This way every byte contributes with 7 bit to the length.

For example: the length field of a 64 byte message simply consists of 1 byte with a 0x40 value. However, if this message's length is 321 byte (2 x 128 + 65), the field becomes two byte long. The value of the first byte is 65 + 128 = 193 according to the LSB first coding. The highest-order bit indicates that there is at least a one byte follow-up. The second byte's value is 2.

Since the protocol limits the length field to 2 bytes, the applications can send a maximum number of 16383 bytes (16 kB) of data in a packet, however, packet size is usually limited by underlying protocols (TCP/IP) to about 1400 bytes.

In Figure 1 . we can see the decoding algorithm of the length field. When the algorithm finishes the value variable will contain the length of the message in bytes.

```
// Pseudo C code
int multiplier = 1;
int value = 0;
do {
  digit = next_byte_from_stream;
  value += (digit & 127) * multiplier;
  multiplier *= 128;
} while ((digit & 128) != 0)
```

5.9 Fixed header notation

In the following sections, in packet descriptions we will mark in a short way the state of the SID, SeqNum, PacketSize, C = Cached, S = Saved and A = AckReq as well as the E = EAX enabled bits.

The value S for the SID field stands for an established session ID, the value N for the SeqNum stands for an arbitrary sequence number and L stands for an arbitrary, correct PacketSize.

For bit-type data the symbol X stands for an arbitrary value, 0 means zero/cleared, and 1 means one/set bit state.

E.g.:

Fixed header								
0 byte	SID				0x0000			
2 byte	SeqNum				0x0001			
4 byte	MessageType				C	S	A	E
	0	0	0	1	0	0	0	0
5 byte	PacketSize				L			

This means the following: Based on the MessageType this is an initial Client » Server CONNECT packet in which the Cached, the Saved, the AckReq, and the EAX flag must always have a fix 0 value, the SID must be 0x0000 and the SequenceNumber must be 0x0001.

The violation of the rules of the flag bits (and in some cases the other fields) MUST always result in the immediate termination of the session. The incorrect operation of any one of the four bits can start a process that leads to a series of communication errors.

An example for this would be the irresponsible use of the AckReq bit: it would be pointless to request ACKNOWLEDGE to a packet that has no MessageID.

6 Session management

6.1 Session establishment

A session can only be initiated by a Client device towards the Server. Connection establishment is comprised of different phases, using special CONNECT type packets.

During connection, the Client is authenticated, and the Server may refuse or accept the request. The current phase of the connection establishment is always marked in the CONNECT packages, in the ConnState field, that is always the first byte of the packet payload (Specialized Header). Possible values for this field include:

Value	Meaning
0x00	Access Denied (Session Closed)
0x01	Initializing New Connection
0x02	Server Authentication Message
0x03	Client Authentication Message
0x04	Access Granted (Session Open)
0x05	Session Paused

If a secure connection is being established, the initialization of the security features is also done during this period. The process is using a four-way handshake based on an ISO/IEC 9798-2 method (Menezes et.al. Handbook of Applied Cryptography p. 402) to provide protection against most attack types and provide mutual authentication.

If no security features are enabled, the connection process is reduced to a simple request-response pair, and no security guarantees are held.

6.1.1 Secure connection establishment

6.1.1.1 Step 1. - Initializing New Connection

If security features are enabled, the Client MUST start a new session by sending an initial CONNECT packet of the following form.

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				0x0000				
2	2	SeqNum				0x0001				
4	1	**MessageType**				**C**	**S**	**A**	**E**
		0	0	0	1	0	0	0	0
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**

0	1	ConnState	0x01
1	2	DeviceType (MSB first)	X
3	4	ModuleID (MSB first)	X
7	8*	ClientInitVector (MSB first)	X

ConnState: MUST be 0x01 – Initializing new connection.

DeviceType: identifies the type of the device that is trying to connect.

Value	Meaning
0x0000	Invalid
0x0001	IRIS.base FW 2.x

ModuleID: The Client's configured ID. The Server uniquely identifies the device using the DeviceType and the ModuleID, and can look up the current device configuration, including security parameters using these values in its local database.

ClientInitVector: This initial vector MUST be random and be half the size of the block/key size of the cipher used. This initial vector will be used as half of the nonce for the EAX method.

All of the fixed header and the packet body MUST be sent as plaintext.

6.1.1.2 Step 2. - Server Authentication Message

The Server needs to look up the security parameters of the Client device using the DeviceType and ModuleID fields of the received packet. If the Server does not have any information regarding the Client device, the session MUST be terminated by the Server (see clause 6.2).

If all is fine, the second step of the secure session establishment is a response from the Server. In this step, the Server MUST fill the SID field of the next packet, by generating an SID (S) for the session (see clause 4.1).

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				0x0001				
		MessageType				**C**	**S**	**A**	**E**
4	1	0	0	0	1	0	0	0	0
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x02
1	4	TimeStamp (MSB first)	X
5	8*	ServerInitVector	X
13	8*	ClientInitVector	X

ConnState: MUST be 0x02 – Server Authentication Message

TimeStamp: The current time in UNIX time format. The Client can use this value to set its clocks.

ServerInitVector: This initial vector MUST be random, and be half the size of the block/key size of the block cipher used. This initial vector will be used as half of the nonce for the EAX method.

ClientInitVector: The very same vector received in step 1.

Please note that the ServerInitVector, and ClientInitVector fields together form a block of data that's size exactly matches the block/key size of the cipher used.

The ServerInitVector, and the ClientInitVector fields together MUST be encrypted using the cipher parameters and encryption key associated to the DeviceType+ModuleID received before, using the ECB mode of the cipher.

6.1.1.3 Step 3. – Client Authentication Message

After the Client receives the Server Authentication Message, the Client MUST decipher the ServerInitVector and the ClientInitVector fields of the message. If the ClientInitVector value received does not match the one sent in step 1, Server authentication failed, and the session MUST be terminated silently.

If the ClientInitVector matches the one sent in step 1, the Client MUST send a Client Authentication Message.

The **fixed header** of the packet MUST include:

Fixed header									
Index	**Size**	**Name**				**Value**			
0	2	SID				S			
2	2	SeqNum				0x0002			
4	1	**MessageType**				**C**	**S**	**A**	**E**
		0	0	0	1	0	0	0	0
5	1	PacketSize				L			

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x03
1	8*	ClientInitVector	X
9	8*	ServerInitVector	X

ConnState: MUST be 0x03 – Client Authentication Message

ClientInitVector: The very same vector sent to the Server in step 1.

ServerInitVector: The very same vector sent to the Client in step 2.

Please note that the two fields containing the initial vectors are swapped compared to the Server Authentication Message. The ServerInitVector and the ClientInitVector fields MUST be encrypted together, using the ECB mode of the cipher.

6.1.1.4 Step 4. – Session establishment

After the Server receives the Client Authentication Message, the Server MUST decipher the ServerInitVector and the ClientInitVector fields of the message. If either of these values do not match the one sent in step 2, Client authentication failed, and the session MUST be terminated silently. If the values are correct, the Server MUST notify the Client about the success.

This packet MUST have EAX enabled.

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				0x0002				
4	1	**MessageType**				**C**	**S**	**A**	**E**	
		0	0	0	1	0	0	0	1	
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x04

ConnState: MUST be 0x04 – Session started successfully.

6.1.1.5 Implementation considerations

If a new connection request arrives from a Client that already has an active OSP session, and the four-way handshake is successful, the previous session MUST be silently terminated.

If a connection request arrives from a Client while a previous connection request from the same Client is pending, the pending session-initiative MUST be silently terminated, and a new connection process SHOULD be started with the same SID that was allocated to the pending session-initiative.

Both parties SHOULD keep a fallback timer set to a reasonable timeout value at the start of every step. If the other party does not respond until the timer is up, the session SHOULD be silently terminated.

6.1.2 Non-secure connection establishment

If no security features are enabled, the connection process boils down to a simple request-response pair, where the authentication is based on the DeviceType + ModuleID only.

6.1.2.1 Step 1. – Initializing New Connection

If no security features are enabled, the Client MUST start a new session by sending an initial CONNECT packet of the following form.

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				0x0000				
2	2	SeqNum				0x0001				
4	1	**MessageType**				**C**	**S**	**A**	**E**	
		0	0	0	1	0	0	0	0	
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x01
1	2	DeviceType (MSB first)	X
3	4	ModuleID (MSB first)	X

ConnState: MUST be 0x01 – Initializing new connection.

DeviceType: See clause 6.1.1.2

ModuleID: See clause 6.1.1.2

6.1.2.2 Step 2. – Session establishment

The Server needs to look up the security parameters of the Client device using the DeviceType and ModuleID fields of the received packet. If the Server does not have any information regarding the Client device, the session MUST be terminated by the Server (see clause 6.2).

If all is fine, the second step of the non-secure session establishment is a response from the Server, setting the SID for the session as well (see clause 4.1).

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				0x0001				
4	1	**MessageType**				**C**	**S**	**A**	**E**
		0	0	0	1	0	0	0	0
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x04
1	4	TimeStamp (MSB first)	X

ConnState: MUST be 0x04 - Session started successfully.

TimeStamp: The current time in UNIX time format. The Client can use this value to set its clocks.

6.1.2.3 Implementation considerations

If a new connection request arrives from a Client that already has an active OSP session, and the connection is successful, the previous session MUST be silently terminated.

The Client SHOULD keep a fallback timer set to a reasonable timeout value at the start of connection establishment. If the Server does not respond to the request until the timer is up, the session SHOULD be silently terminated.

6.2 Session termination

If a session is not terminated silently, sessions SHOULD be closed using CONNECT packets, where the ConnState field is set to 0x00 – Session Terminated. A closing CONNECT packet can be sent by any party, at any point during a session, and MUST result in the immediate closing of the session, so both the sending, and receiving party MUST cease sending additional packets, and MUST drop all incoming packets that have the SID of the inactive session.

These packets MUST obey all rules regarding SID, SeqNum, and MessageSize.

If sent in a secure connection, this packet MUST have EAX enabled.

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				N				
		MessageType				**C**	**S**	**A**	**E**	
4	1	0	0	0	1	0	0	0	X	
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x00

ConnState: MUST be 0x00 - Session terminated.

6.3 Session pausing

For sensors and devices that need to send data to the Server with long sleep periods, OSP offers the possibility to send a session into dormant state using a special CONNECT packet, and then re-activate the session later without re-initializing it. Only the Client MAY send a session into dormant state.

If sent in a secure connection, this packet MUST have EAX enabled.

The **fixed header** of the packet MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				N				
		MessageType				**C**	**S**	**A**	**E**	
4	1	0	0	0	1	0	0	0	X	
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	ConnState	0x05

ConnState: MUST be 0x05 - Session paused.

In dormant state, lower-layer network sessions (if any) MAY be disconnected, and the Server MUST NOT send any further packets to the Client, until the session is re-activated. If the Client sends any correct packet in a dormant session (where the SeqNum is bigger than the SeqNum of the CONNECT packet used to pause the session), the session is re-activated.

It is RECOMMENDED to re-activate sessions by sending a PINGREQ to the Server (see clause 8).

If a secure connection is being re-activated, the packet used for re-activation MUST have EAX enabled.

6.4 Session monitoring

If no lower layer service is available, to monitor the integrity of a connection, OSP offers an application level ping service. Any party MAY send PINGREQ packets in an active session, to which the other party MUST respond with exactly one PINGRESP packet. Unexpected PINGRESP packets MUST be ignored.

If the PINGREQ-sending party does not receive a PINGRESP within a specific timeframe (depending on implementation), it may indicate an error of the connection, and it is RECOMMENDED to cease the sending of DATA, FIRMWARE and COMMAND packets through the unreliable connection, until the issue is resolved.

If using non-reliable lower layer protocols, the PINGREQ-sending party MAY send more than one PINGREQ packet before assuming a broken connection. If a broken connection is detected, it is RECOMMENDED to re-initialize the lower-layer network protocol stack only. If no lower-layer error is detected during re-initialization, it is RECOMMENDED to start a new OSP session.

Neither the PINGREQ, nor the PINGRESP packets have body.

If sent in a secure connection, it is RECOMMENDED to enable EAX for these packets.

The **fixed header** of the PINGREQ packets MUST include:

Fixed header											
Index	**Size**	**Name**				**Value**					
0	2	SID				S					
2	2	SeqNum				N					
		MessageType				**C**	**S**	**A**	**E**		
4	1	0	1	0	0	0	0	0	X		
5	1	PacketSize				L					

The **fixed header** of the PINGRESP packets MUST include:

Fixed header			
Index	**Size**	**Name**	**Value**

0	2	SID				S			
2	2	SeqNum				N			
		MessageType				**C**	**S**	**A**	**E**
4	1	0	1	0	1	0	0	0	X
5	1	PacketSize				L			

7 Client configuration, commands

To configure a Client, send commands or other data, OSP offers the COMMAND packets. The packet contains the commands in a textual or binary form.

Only the Server MAY initiate a command request. To every Server-originated COMMAND request packet the Client MUST respond with exactly one COMMAND response packet. The Server MUST NOT send additional requests while a previous request is pending. If a command request is received by the Client while a previous command is being executed, the Client MUST respond with an ExitCode of 0x02 (BUSY), and an empty ResponseString.

If the Server does not receive a response to a request in a specified (implementation dependent) timeframe, the request SHOULD be considered failed.

The actual format of the command scripts and responses is out of the scope of this specification. It's the Server's responsibility to send commands that the Client can understand. These commands should be found in the documentation of the specific device that implements the OSP.

The COMMAND request packets always contain an in-flight-unique command identifier, and COMMAND responses MUST have the same identifier set, so responses can be unambiguously linked to the requests.

If sent in a secure connection, these packets MUST have EAX enabled.

The **fixed header** of the COMMAND packets MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				N				
4	1	**MessageType**				**C**	**S**	**A**	**E**	
		0	0	1	0	0	0	0	X	
5	1	PacketSize				L				

7.1 Server » Client

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	CommandID	X
1	N	CommandScript (string)	-

CommandID: The command's in-flight-unique identifier. It is important to define it, because this way the responses to consecutive commands can be identified unambiguously.

Script: The command and it's parameters in a textual or binary form.

7.2 Client » Server

The Client's response to one of the commands sent by the Server. It is REQUIRED to send exactly one response to every command request.

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	CommandID	X
1	1	ExitCode	X
2	N	CommandResponse (string)	-

CommandID: The command's in-flight-unique identifier. The response to the command request MUST contain the same CommandID as the request. If the Server receives a COMMAND packet with an invalid ID, the packet MUST be ignored.

ExitCode: The command's return code that MAY depend on the device implementation.

Value	Meaning
0x00	Failed (Generic Error)
0x01	Success
0x02	Busy
0x02-0xFF	Application dependent other error code

Response: The Client's answer to the command. (Can also be an empty string)

8 Client firmware updating

To update the firmware of a Client, the OSP offers the FIRMWARE packets. Using these packets, the Client can identify which firmware to download with its name and can request it in arbitrarily sized chunks from the Server.

The way a firmware updating process is started, what size or internal structure each firmware chunk has, and how each firmware chunk's integrity is checked are out of the scope of this specification.

The firmware is downloaded in a request-response manner, similar to commands, initiated by the Client. To every Client-originated FIRMWARE request packet, the Server MUST respond with exactly one FIRMWARE response packet. A Client MUST NOT send additional requests while a previous request is pending. If the Client does not receive a response to a request within a specified (implementation dependent) timeframe, the request SHOULD be considered failed.

If the Client receives a FIRMWARE packet not currently requested, the packet MUST be ignored.

If the Client requests a firmware version or a chunk that doesn't exist, the Server MUST respond with a FIRMWARE packet with empty ChunkData.

If sent in a secure connection, these packets MUST have EAX enabled.

The **fixed header** of the FIRMWARE packets MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				N				
4	1	**MessageType**				**C**	**S**	**A**	**E**
		0	1	1	0	0	0	0	X
5	1	PacketSize				L				

8.1 Client » Server

The Client requests a specific chunk of a specific firmware. The packet contains the globally explicit name (version) of the firmware and the chunk.

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	2	ChunkID	X
2	20	FirmwareName (string)	-

ChunkID: The ID of a firmware chunk. It's a 2-byte, unsigned value, allowing up to 65535 chunks.

FirmwareName: A 20 character long text field that identifies the firmware (version).

8.2 Server » Client

To every correct FIRMWARE request packet from a Client, the Server MUST respond with exactly one FIRMWARE response, containing the firmware chunk data, or an error indication, if the requested chunk could not be found.

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	2	ChunkID	X
2	20	FirmwareName (string)	-
22	X	ChunkData (string)	-

ChunkID: The ID of a firmware chunk. It's a 2-byte, unsigned value, allowing up to 65535 chunks.

FirmwareName: A 20 character long text field that identifies the firmware (version).

ChunkData: The binary of the firmware chunk. The size of the data is not defined by the OSP, but is limited by the PacketSize to ~16kB (usually further limited by underlying protocol limitations, e.g. in the case of TCP/IP to ~1.4kB) (see clause 4.8). If the server could not find the firmware chunk, this field MUST be empty.

9 Sending data

9.1 DATA packets

To send data to Server, the OSP offers the DATA packets. **DATA packets can only be generated by the Client and sent to the Server.** If a Client receives a DATA packet, the packet MUST be ignored. Data packets can be cached or archived (saved to non-volatile memory) on the Client side. It is also possible to request acknowledgements from the Server, if reliable transmission cannot be guaranteed by lower-layer protocols.

If sent in a secure connection, it is RECOMMENDED to enable EAX for these packets.

The **fixed header** of the DATA packets MUST include:

Fixed header									
Index	**Size**	**Name**				**Value**			
0	2	SID				S			
2	2	SeqNum				N			
		MessageType				**C**	**S**	**A**	**E**
4	1	1	0	0	0	X	X	X	X
5	1	PacketSize				L			

The DATA packet is the only packet type, where the C, S, and A flags MAY be set. (see clause 4.)

DATA packets have a Specialized Header at the start of the payload, comprising the MessageID, and the DataType fields. After the Specialized Header, the DATA packet contains an arbitrary-sized payload.

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	MessageID	X
1	2	DataType	X
3	X	DataPayload (string)	-

MessageID: The in-flight-unique identifier generated by the Client to the DATA packet. The minimum value of the MessageID is 0, while the maximum value is device-dependent. The maximum value of the MessageID for a device should be chosen so that the device can hold at least this many of the biggest possible DATA packet in its cache.

With each new DATA packet sent, the MessageID MUST be incremented by 1, until it reaches its maximum value, after which the MessageID must fold back to 0. This way the designation of the MessageIDs is continuous, which means that the Server can detect any data drop-out and can send a RESEND request to the Client with the missing MessageID within an active OSP session (no RESEND requests can be sent for DATA packets from a previous session!).

DataType: This parameter defines the structure and content of the Payload. It's a 2-byte, unsigned value.

Reserved values:

Value	Meaning
0x0000*	An error message on the RESEND packet, see clause 12.
0x0001-0x009	Reserved for development and testing.
0x000A-	Device specific data packet.

***DataType 0x0000:** These packets can be sent from the Client to the Server if the Server sent a RESEND packet to request a DATA packet that is unavailable. In this case the Server acknowledges that the ID is invalid and moves on.

Payload: A data content that is specified by the DataType, its length and its structure is out of the scope of this specification. The detailed description of the defined DataTypes can be found in the appendix.

9.2 RESEND packets

The RESEND service works in a request-response manner, initiated by the Server. To every Server-originated RESEND request packet, the Client MUST respond with exactly one DATA packet from its cache, with the MessageID specified by the request. The Server MAY send additional requests while a previous request is pending. If the Server does not receive a response to a request within a specified (implementation dependent) timeframe, the request SHOULD be considered failed.

If the Client receives a RESEND request for a DATA packet that is no longer in its cache, the Client MUST respond with a DATA packet stub, where the DataType field is 0x0000, and the Payload is empty.

As the RESEND requests may only be sent by the Server, any RESEND requests received by the Server MUST be ignored.

If sent in a secure connection, it is RECOMMENDED to enable EAX for these packets.

The **fixed header** of the RESEND packets MUST include:

Fixed header										
Index	**Size**	**Name**				**Value**				
0	2	SID				S				
2	2	SeqNum				N				
4	1	**MessageType**				**C**	**S**	**A**	**E**	
		0	1	1	1	0	0	0	X	
5	1	PacketSize				L				

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	MessageID	X

MessageID: The identification of the DATA packet requested to be resent by the Client.

9.3 ACKNOWLEDGE packets

If the underlying network protocol stack is unreliable, OSP offers an ACKNOWLEDGE mechanism, that can be enabled for select DATA packets. If the Client sets the A flag in a DATA packet, the Server MUST respond to this packet with exactly one ACKNOWLEDGE packet, containing the in-flight-unique MessageID of the DATA packet acknowledged.

If the Client does not receive an ACKNOWLEDGE response to a DATA packet sent to Server with the A flag set in a specified (device dependent) timeframe, the Client MUST automatically send the DATA packet again. A DATA packet of which the A flag is set MUST NOT be deleted from the Client's cache until an ACKNOWLEDGE packet containing it's MessageID is received from the Server, or the OSP session is closed/terminated.

As ACKNOWLEDGE packets are only allowed in the Server » Client direction, any ACKNOWLEDGE packets received by the Server MUST be ignored.

If sent in a secure connection, it is RECOMMENDED to enable EAX for these packets.

The **fixed header** of the ACKNOWLEDGE packets MUST include:

Fixed header									
Index	**Size**	**Name**				**Value**			
0	2	SID				S			
2	2	SeqNum				N			
4	1	**MessageType**				**C**	**S**	**A**	**E**
		0	0	1	1	0	0	0	X
5	1	PacketSize				L			

Packet body:

Packet body			
Index	**Size**	**Name**	**Value**
0	1	MessageID	X