constant robotics

Network Transport Protocol v5.0

specification

clib

www.constantrobotics.com

CONTENTS

DOCUMENT VERSIONS

Table 1 – Document versions.

Version	Release date	What has changed
3.0	13.07.2020	Specification of Network Transport Protocol version 3.0.
4.0	28.01.2021	Specification of Network Transport Protocol version 4.0.
5.0	02.08.2021	Specification of Network Transport Protocol version 5.0.

PROTOCOL VERSIONS

Table 2 – Protocol versions.

Version	Release date	What's new
3.0	13.07.2020	1. The principle of data exchange has been changed from requesting lost data to confirming received data.
4.0	28.01.2021	1. The principle of data exchange has been changed from confirming received data to requesting lost data.
5.0	02.08.2021	1. A confirmation mechanism for accepted data has been added. 2. New types of information messages have been added.

DESCRIPTION

The **Network Transport Protocol** version **5.0** (hereafter referred to as the protocol) is designed for the transmission of any data over a network via UDP packets. According to the OSI classification, the protocol belongs to the session or application layer. The protocol is a layer over the UDP protocol and provides reliable data delivery of up to 4 169 727 bytes. The protocol is designed for reliable data transmission over radio links and provides data exchange in the presence of UDP packet loss. The protocol allows up to 16 virtual data channels per UDP port and includes an acknowledgement mechanism for received data. The protocol has a minimum number of service bytes (6 bytes for data packets) added to the data being transmitted.

PROTOCOL PARAMETERS

Table 3 – Protocol parameters.

Parameter	Value
OSI layer	The protocol is a layer over the UDP protocol. The OSI layer is the session or application layer.
Packet loss processing	The sender divides the volume of data to be transmitted into individual packets (sections) and sends them sequentially. When the receiver detects a lost packet (when a missing packet is detected), it sends a request to the sender to resend the lost data. The sender, having received the request to resend the lost data, sends it (lost data) firstly.
Confirmation of received data	When the receiver receives all packets relating to the same logically linked data, it sends an acknowledgement packet of the received data to the sender.
Maximum data packet size	1024 bytes of which 1018 bytes of data and 6 service bytes.
Maximum size of data to be transmitted	4 169 727 bytes (e.g. video frame data size).
Adapting to channel capacity	Packets should be sent at intervals appropriate to the communication channel bandwidth.

DATA EXCHANGE PRINCIPLE

The protocol defines four types of data packets: **DATA** – packets with data to be transmitted (with or without acknowledgement sign), **MARKER** – packets completing the transmission of logically related data, **LOST_DATA_REQUEST** – request to resend lost data and **CONFIRM** – acknowledgement packet of received data. The principle of data exchange using the protocol is as follows: the sender divides the transmitted data into sections of a fixed size of 1018 bytes, except for the last section, which may be smaller than 1018 bytes (unless the volume of data to be transmitted is divided by exactly 1018 bytes). Logically linked data (e.g. one video frame data) is assigned a unique identifier that allows the receiver to determine when new data has been received. A Data ID is included in the each **DATA** packet being transmitted. Each data section (with a maximum size of 1018 bytes) is packed into a **DATA** packet – a packet header of 6 bytes is added to the data being transmitted. Generated packets with a maximum size of 1024 bytes are subsequently sent by the sender to the specified IP address via the specified UDP port. After all data packets have been sent, the sender sends two **MARKER** packets (1 byte size each), signaling to the receiver that it has finished sending data. The receiver receives **DATA** packets, extracts a useful amount of data from them (maximum 1018 bytes) and copy them in the input data buffer. When a missing packet is detected (each packet has a sequential number), the receiver generates a **LOST_DATA_REQUEST** to resend the lost data and sends it to the sender. Lost packets are detected based on their sequence number (packet ID) – if there is a gap in the sequence of received packet numbers, this indicates that there are lost packets (one or more). The receiver also checks for lost **DATA** packets when processing **MARKER** packets – there are lost packets if the number of packets received (packets with the same data ID) is less than the total number of packets that should have been transmitted. The sender when receiving a **LOST_DATA_REQUEST** will resend the lost packets firstly. The protocol allows the transmitted data to be divided into individual virtual channels (up to 16 channels – logical ports) for sending several different data types over a single UDP port with separation in receiver's side. If data was transmitted with **DATA** packets with acknowledgement sign, the receiver sends **CONFIRM** packet – acknowledgement packet of received data for sender after receiving logically connected data (after receiving all **DATA** packets belonging to the same data ID, e.g. one video frame).

Figure 1 shows the data being transferred (pink and purple). Each of logically related data is assigned a different identifier (**data_ID == 0** and **data_ID == 1** in figure 1). The Data ID are assigned sequentially starting with 0, with the next data to be transmitted being assigned the identifier 0 again once the maximum value of 15 is reached (0, 1, 2, …, 15, 0, 1 …). To avoid confusion with the same Data ID, there are additional unique identifiers for each data_ID value as part of the **DATA** packets (part_ID). The protocol requires the sender to have two buffers: a packets buffer and a send data buffer. The send data buffer contains the packets that the sending stream sends to the receiver. If lost data needs to be resent, the sender copies the relevant packets from the packets buffer to the send data buffer.

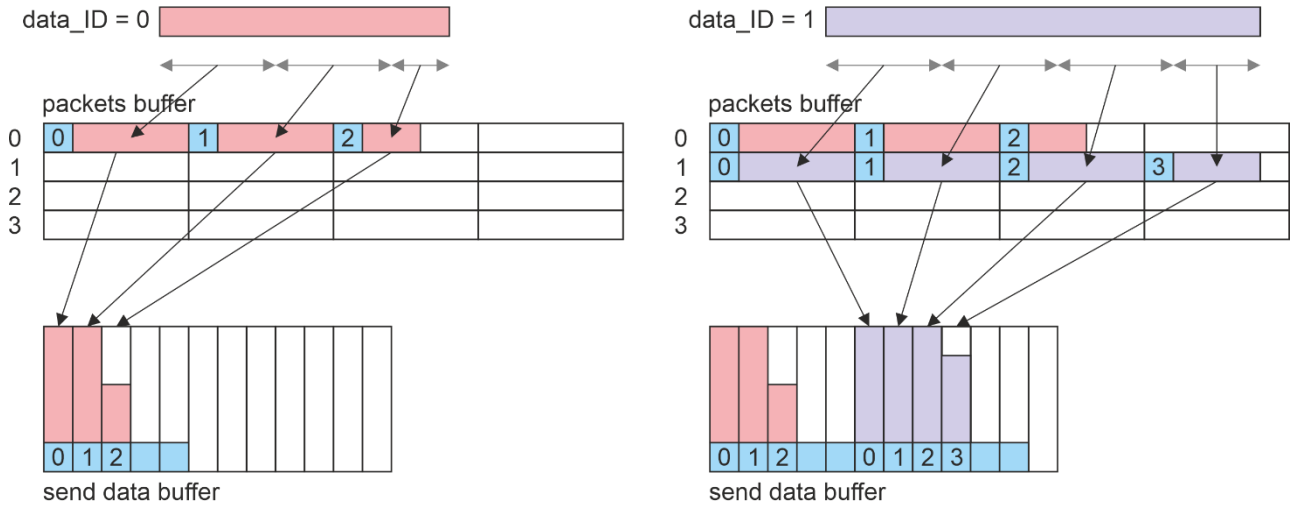

Figure 1 – Principle of filling the packets buffer and the send data buffer.

When the data is sent, it is divided into consecutive sections of 1018 bytes except for the last section which may be smaller (as shown in Figure 1). For each piece of data, a **DATA** packet is

generated and placed in the packets buffer. The packet buffer is an array of 4096 cells (the maximum number of packets for a single logical data transfer) with a size of 1024 bytes (the maximum size of a transmitted packet) for each of the possible packet ID (16 buffers of 4096 packets in total). In Figure 1, for ease of explanation of protocol operation, the maximum number of **DATA** packets for data transmitted is 4 and the maximum number of data ID variants is also 4 (Figure 1 shows 4 buffers of 4 data packets each). The packets generated from the transmitted data are placed in the corresponding cells of the packet buffer (according to the data_ID and packet number, starting from 0). Figure 1 shows the adding of two amounts of data with data_ID == 0 and data_ID == 1 to the packets buffer. The first data is assigned data_ID == 0. This data is divided into 3 **DATA** packets and placed in the packet buffer. The following data is assigned data_ID == 1 and is divided into 4 **DATA** packets. When data is added to the packets buffer, the generated packets are added to the send data buffer. Once all required **DATA** packets have been added, two **MARKER** packets (1 byte size each) are added to the send data buffer to signal the receiver that the sender has completed sending the data. The send data buffer is required by the sending stream, which sequentially reads **DATA** packets from it and sends them to the receiver. When a request is received to resend lost data, the relevant packets (specified in the **LOST_DATA_REQUEST**) will be re-posted to the send data buffer from the packets buffer. The packets buffer can store up to 15 previous amounts of data to provide operation in case of high link time delays.

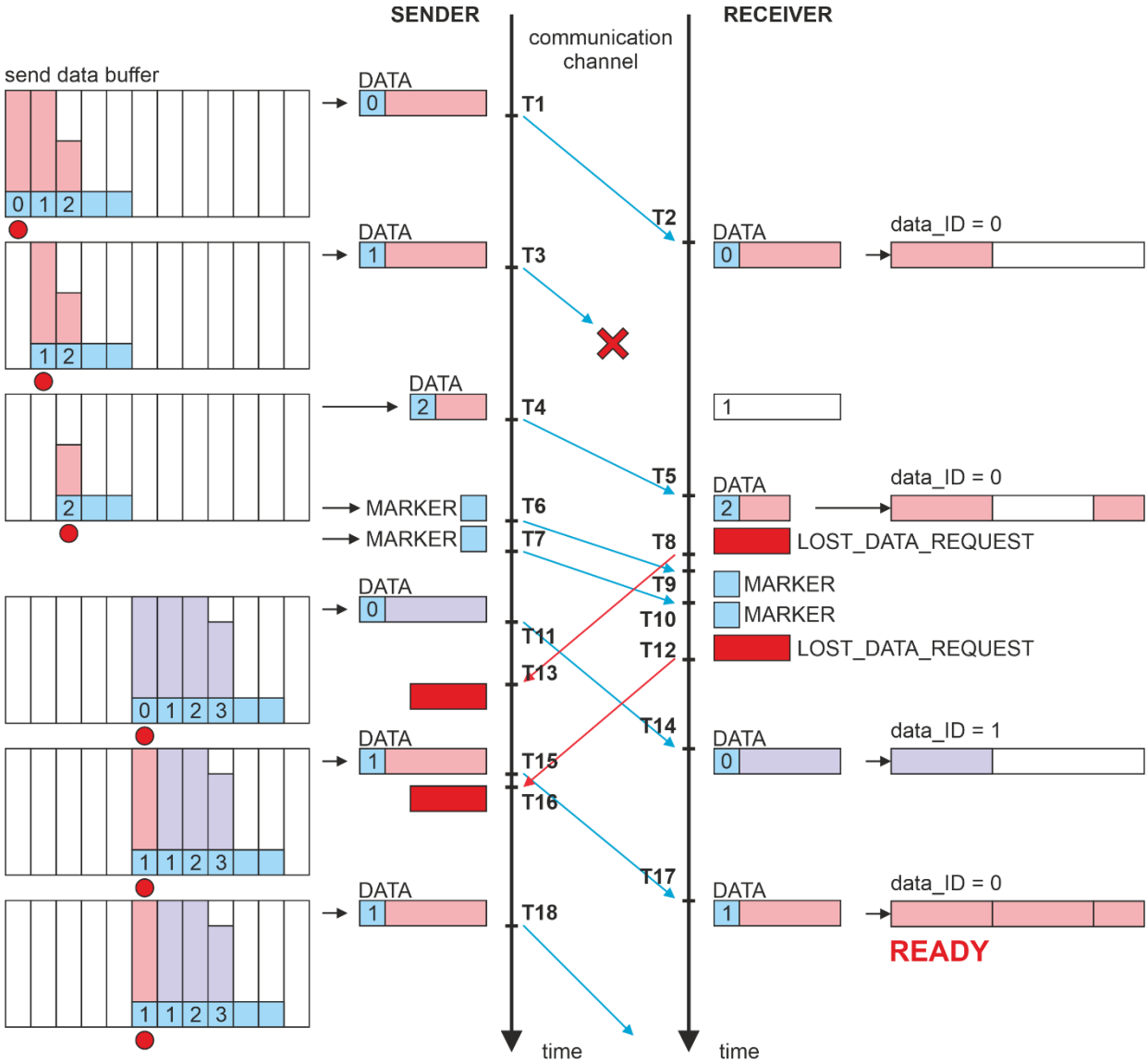

Figure 2 – The principle of information exchange.

Figure 2 shows the principle of information exchange and illustrates the principle of reading **DATA** packets from the data sending buffer and sending them. The principle of data exchange with **DATA** packets with acknowledgement sign is similar, except that the receiver sends a **CONFIRM**

packet on receipt of all **DATA** packets relating to the same logically related data. Figure 2 shows the state of the send data buffer when sending **DATA** packets. The data send buffer size is 65536 packets. A large outgoing packet buffer size is required to provide stable data transmission in an unstable transmission environment. In Figure 2, for ease of explanation, the size of the send data buffer is equal to 12 **DATA** packets. The red dot at the bottom of the corresponding buffer cells indicates the current position for reading the packet to be sent. After sending the next packet, the sending thread increases the index by one (goes to the next packet). If the send thread sees a cell with no data to send, it expects new data to be added to the buffer. Consider the example shown in Figure 2.

At time **T1**, the send thread reads a **DATA** packet with index 0 from the send data buffer and sends it out. The packet takes some time to transmit and at time **T2** the sent packet with index 0 is received by the receiver. The receiver processes the received packet and places the data transmitted in it into the input data buffer starting at the appropriate position. The address of the first byte to copy the packet data to the input buffer is calculated as follows:

$$data\_position = 1018 * packet\_ID \quad (1)$$

where: $packet\_ID$ – a packet index between 0 and 4095 (the maximum packet index value for the data to be transmitted is calculated as $data\_size/1018$).

After sending a packet with index 0, the sender jumps to the next cell of the send buffer and clears the cell with the already sent packet. After a certain time interval (at time point **T3**) the sender sends the next **DATA** packet – packet with index **1**. It then moves on to the next packet in the send data buffer. The time interval in microseconds between packets (between the start of sending the previous packet and the current packet) depends on the size of the previous packet transmitted and is calculated as follows:

$$interval\_mks = \frac{packet\_size * 8}{channel\_bandwidth\_Mbps} \quad (2)$$

where: $interval\_mks$ – time interval between sending packets in microseconds; $packet\_size$ – the size of the previous packet transmitted (1024 bytes or less); $channel\_bandwidth\_Mbps$ – channel bandwidth in megabits per second (determined by the sender).

Assume that the package with index **1** has been lost. At time **T4**, sender sends packet with index **2** (last **DATA** packet for data with index **data_ID == 0**). At time **T5** a **DATA** packet with index **2** is received by the receiver. The receiver processes the incoming packet and copies the data it contains into the appropriate area of the input buffer. As you can see from the example, the lost package is index **1**. The receiver determines that a packet has been lost on the basis that there is a gap in the IDs of successive received packets. At this time, the sender at time moments **T6** and **T7** has sent two final **MARKER** packets of **1** byte in size with a minimum time interval between them. Immediately after packet loss is detected, the receiver at time **T8** generates a lost data request **LOST_DATA_REQUEST**, which includes the index of the lost packets (index 1) and the ID of the data to which they relate (index **0**). A lost data request is sent to the receiver. While the **LOST_DATA_REQUEST** sends in the example in figure 2 the **data_ID == 1** is added to the send data buffer and the sender sends a **DATA** packet with **data_ID == 1** at time **T11**. By this time the receiver has already received two **MARKER** packets. When receiving **MARKER** packets, the receiver has detected that there are lost packets. The presence of lost packets in this case is indicated by the fact that the total number of received packets is less than or equal to the maximum packet index for the **data_ID == 0** (in this example it is index **2**, transmitted in the **DATA** packet). The receiver generates a **LOST_DATA_REQUEST** and sends it at time **T12**. At time **T13** the sender receives a **LOST_DATA_REQUEST** containing a lost packet index **1** (only one in this example). **Data_ID == 0** is included in **LOST_DATA_REQUEST**. The sender copies the relevant packet from the packet buffer to the send buffer. Copying is performed so that the packets requested for retransmission are first in the sending queue for the sending stream. To do this, the receiver sets the position for sending back the required number of packets (after time moment **T11** the sender has moved to the next packet **data_ID == 1** and **packet_ID == 1**) and copies the packet data for sending. At time **T14**, the receiver receives a packet with index **0 data_ID == 1** and stacks it in the receive buffer corresponding to data index **1**, while the buffer for data index **0** remains empty. At time **T15** the sender re-sends the lost packet with index **1** for **data_ID == 0**. At time **T16**, the sender receives a second **LOST_DATA_REQUEST** generated by the receiver after processing the **MARKER** packets. **LOST_DATA_REQUEST** packets are generated each time a **MARKER** packet is processed in case of

lost data. In this example, for simplicity, only one **LOST_DATA_REQUEST** packet is generated after processing two **MARKER** packets. The sender, as in the previous case, adds the corresponding packet with index **1** to the send buffer. At time **T17** the receiver receives the missing data packet **data_ID == 0** and copies the data contained in the **DATA** packet into the receive buffer. The **data_ID == 0** is then fully collected and the receiver outputs the data ready signal. The sender sends the missing packet (**DATA** packet with index 1) again at time **T18**, but it will be discarded by the receiver. The sender will then revert back to sending **data_ID == 1** data packets. In this way, the protocol performs monitoring and retransmission of lost packets to provide reliable data transmission. If **DATA** packets contain an indication of the need for reception acknowledgement, the receiver sends a **CONFIRM** packet to the sender after the relevant data has been collected on the receiving side.

In addition to the principle described above, the protocol has the ability to work in one-way communication channels. In order to ensure reliable data transmission under conditions of packet loss on one-way links (with data transmission only to one side), the protocol has the ability to add packets with the same data ID to the send data buffer several times (the number of repetitions is determined by the user).

DATA PACKET FORMAT

DATA packets are required for data transmission. DATA packets have a maximum size of 1024 bytes of which 6 header bytes and 1018 data bytes. The minimum size of a DATA packet is 7 bytes (1 byte of data). DATA packets have the following format:

bit	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
field	0	0 or 1	0	1	logic_port				data_ID				part_ID			
bit	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
field	0	0	0	0	packet_ID											
bit	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
field	0	0	0	0	max_packet_ID											
bit	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
field	data[0]								data[1]							
bit									**8184**	**8185**	**8186**	**8187**	**8188**	**8189**	**8190**	**8191**
field	**…**								data[1017]							

Table 4 – DATA packet fields.

Field	Value	Description
Header	0x1 or 0x5	The packet header occupies the first 4 bits of the first byte of the packet. It is fixed at **0x1** for **DATA** packets without acknowledgement sigh or **0x5** with acknowledgement sign.
logic_port	from 0 (0x0) to 15 (0xF)	The logical port number is the last 4 bits of the first byte of the packet. The data being transmitted can be divided into 16 virtual ports (streams). For example, data can be split by destination for transmission over a single UDP port.
data_ID	from 0 (0x0) to 15 (0xF)	The identifier of the data being transmitted. When data (e.g. video frame data) is transmitted, the sender sequentially assigns a new identifier to the data. When the value 15 is reached, the next ID will be 0 (0, 1, … 15, 0, 1, …).
part_ID	from 0 (0x0) to 15 (0xF)	Incremental counter for data_ID. The counter value is formed as follows: for each data_ID value each time the part_ID counter value is incremented by 1. If the counter value is 15 then the next value will be 0 (0, 1, … 15, 0, 1, …). For example, data has been assigned index 0 (data_ID == 0). The part_ID value for them will be 0 after time the next data will also have ID 0, but the counter value of part_ID must be 1. The counter is necessary for the receiver to decide that new data has arrived and the corresponding input buffer must be reset (according to the data_ID) before copying the new data.

packet_ID **(big endian)**	from 0 (0x000) to 4095 (0xFFF)	Packet ID (12 bits). The data to be sent is divided into packets and each packet is sequentially assigned an identifier from 0 (0x000) to 4095 (0xFFF). The identifier value is sent in **big endian format**.
max_packet_ID **(big endian)**	from 0 (0x000) to 4095 (0xFFF)	Maximum packet identifier for data to be sent (last packet identifier). If size of data to be sent is less than 1018 bytes then all data will be sent in one packet. Hence, packet_ID and max_packet_ID fields will be the same and equal to 0 (0x000). The identifier value is sent in **big endian format**.
data[n]	any	Data to be transmitted. The maximum size of data included in a DATA packet is 1018 bytes.

MARKER PACKET FORMAT

MARKER packets are required to signal to the receiver that the data transmission, with the corresponding identifier (data_ID) is complete. When a MARKER packet is received, the receiver is able to check if all packets have been received and, if a packet loss is detected, generates a LOST_DATA_REQUEST request to retransmit the data. After transmitting all DATA packets corresponding to a specific data ID (data_ID), the sender sends two MARKER packets in sequence with the shortest possible interval between them. The MARKER packets are 1 byte long in the following format:

bit	0	1	2	3	4	5	6	7
field	0	0	1	0	data_ID			

Table 5 – MARKER packet fields.

Field	Value	Description
Header	0x2	The packet header occupies the first 4 bits of the first byte of the packet. It has a fixed value.
data_ID	from 0 (0x0) to 15 (0xF)	The ID of the data to which the MARKER packets relate.

LOST_DATA_REQUEST PACKET FORMAT

LOST_DATA_REQUEST packets are intended to request retransmission of lost DATA packets. The LOST_DATA_REQUEST packet has a minimum size of 4 bytes and a maximum size of 1024 bytes. The packet includes the lost packet IDs that the receiver has identified. In total LOST_DATA_REQUEST packet can include up to 511 DATA packet IDs for retransmission by the sender. LOST_DATA_REQUEST packets have the following format:

bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
field	0	0	1	1	0	0	0	0	data_ID				part_ID			
bit	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
field	0	0	0	0	lost_packet_ID #1											
bit	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
field	0	0	0	0	lost_packet_ID #2											
bit	… up to 1024 bytes															
field	… up to 1024 bytes															

Table 6 – LOST_DATA_REQUEST packet fields.

Field	Value	Description
Header	0x30	The packet header occupies the first byte of the packet. It has a fixed value.
data_ID	from 0 (0x0) to 15 (0xF)	Data identifier to which the packets requested for retransmission correspond.

part_ID	from 0 (0x0) to 15 (0xF)	Incremental counter for data_ID, transmitted with DATA packets.
lost_packet_ID **(big endian)**	from 0 (0x000) to 4095 (0xFFF)	Identifier of a lost packet for retransmission (12 bits). The identifier value is sent in **big endian format**.

CONFIRM PACKET FORMAT

CONFIRM packets are required to acknowledge receipt of data. After receiving all DATA packets belonging to the same logically linked data (data united by one data_ID), the receiver sends a CONFIRM packet to the sender. CONFIRM packets are only sent if the DATA packets contain an acknowledgement to receive (DATA packet header is 0x5). CONFIRM packets are 1 byte in the following format:

bit	0	1	2	3	4	5	6	7
field	0	1	0	0	data_ID			

Table 7 – CONFIRM packet format.

Field	Value	Description
Header	0x4	The packet header occupies the first 4 bits of the first byte of the packet. It has a fixed value.
data_ID	from 0 (0x0) to 15 (0xF)	The identifier of the data whose reception is confirmed.