BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Electronics and Business

Reverse Engineering Loxone Link

By: Stefan Oberpeilsteiner
Student Number: 1610255090

Supervisor 1: Mathias Ballner, MSc

Vienna, 09.06.2018

Declaration of Authenticity

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz/ Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.”

Vienna, 09.06.2018	[illegible]
Place, Date	Signature

Kurzfassung

Diese Bachelorarbeit hat es zum Ziel das Loxone Link Protokoll zu verstehen und den Paketaufbau zu dokumentieren. Der Loxone Link findet Verwendung in Smart Home Produkten der österreichischen Firma Loxone Electronics und basiert auf dem CAN Bus.
Es existieren bereits einige Analysen, jedoch gibt es immer noch viele unbekannte Pakete.
Der Umfang der Arbeit wurde in dem Sinne eingegrenzt, dass nur die Kommunikation zwischen dem Loxone Miniserver und einer DMX Extension betrachtet wurde.

Es kam eine empirische Methodik zu Anwendung. Ein Testaufbau wurde erarbeitet der es erlaubt zuverlässig und reproduzierbar Stimuli von außen auf die Eingänge des Systems zu beaufschlagen und anschließend die Ausgänge zu messen. Dabei wurde auch der Datenverkehr auf dem Bus aufgezeichnet und ausgewertet. Damit war es möglich Rückschlüsse auf die Bedeutung der einzelnen Datenbytes zu ziehen.

Die Ergebnisse sind vertrauenerweckend, weil sie sich gut mit den Resultaten anderer Arbeiten decken.

Schlagwörter: Smarthome, Loxone, CAN, DMX, Miniserver

Abstract

This thesis aims to reverse engineer, understand and document the packet layout of the Loxone Link protocol which is based on the CAN Bus. Loxone Link interconnects Smart Home components of Austrian company Loxone Electronics. There are already some parts that have been analyzed but there is still a lot to be learnt. The scope was limited to analyzing the communication between the Loxone Miniserver and one Loxone DMX Extension.

An empirical method was developed that allows to reliably and repeatably stimulate the systems input and measure the systems outputs. The bus traffic was then sniffed and analyzed to gain an understanding of the meaning of all data bytes.

The findings of this thesis were compared to earlier findings of other researchers. The results are confidence inspiring because they show a lot of similarities with other research.

Keywords: Smarthome, Loxone, CAN, DMX, Miniserver

Acknowledgements

First, I would like to thank my thesis supervisor Mr. Mathias Ballner, MSc of the Integrative Urban Development – Smart City Master’s Degree Program at the USA Technikum Wien.
His office door was always open whenever I had a question about my writing.

I would also like to thank FH-Prof. Mag. DI Dr. Friedrich Praus of the Smart Homes and Assistive Technologies department at the USA Technikum Wien for giving me access to their Loxone Hardware

Finally, I want to express my deepest gratitude to my parents and my partner for providing me with support while writing this thesis.

Table of Contents

1 Introduction

1.1 Motivation

The power of bits and bytes is fascinating - they control so many things in our modern world:

You wake up in the morning by the sound of your alarm clock whose ringing is triggered when the timer-counter value matches its preprogrammed limit.
Then you get up and turn on the lights of your smart home. The brightness and color temperature are of course precisely crafted for a perfect morning mood.
You wash your face with warm water that has been heated a few hours ago when your boiler received a short signal from the power company to consume the cheaper excess energy during the night hours.
Afterwards you check the weather on your smartphone, so you can decide what to wear. The amount of data that has been processed to provide you with this information is awe-inspiring.
By the time you leave your house you have already used a tremendous number of bits and bytes to your advantage.

A developer of a product which must communicate with another product needs to know which bit is responsible for which action. This is often an easy task as the relevant protocols are in many cases documented by the manufacturer.
But in some cases, the Manufacturer chooses to keep this a secret. This may have many reasons. Maybe he is simply not bothering to document the information in any sensible way. Or the Manufacturer hopes to gain an advantage by locking competition out of the market.

Whatever the reason might be, it is a very interesting challenge to find out what the bits and bytes do and mean. Out of that challenge arises this thesis aim to present ways how to capture, analyze and dissect packets sent between Smart Home products by Austrian Company Loxone Electronics.

1.2 Subject Area

This thesis focuses on Smart Home products by the Austrian company Loxone Electronics. The heart of the Loxone Smart Home is the so called "Miniserver". This is how Loxone describes the Miniserver in their e-commerce shop: [1]

> *The Miniserver lays the foundation for every installation and comes with our free software and mobile apps for convenient configuration and control.*
>
> *In a Loxone Smart Home, everything is managed from central nerve centre: the Miniserver. From switches and simple sensors, to lighting, heating, and even complex systems such as solar, everything is brought together centrally. This allows for simpler control and more intelligent automation of your home.*

The following chapters will explain the underlying technologies to better understand the rest of the work. Loxone Link is based on the CAN Bus, the Extension used for testing outputs DMX-512 and supports RDM.

1.2.1 Loxone Smart Home Topology

The Miniserver is typically installed in the central switch cabinet of a house.
The Miniserver has very basic Input and Output options like relays to switch loads and circuitry to measure analog voltages. To extend the capabilities of an installation Loxone offer so called" Extensions" with varying functionalities. These Extensions are connected to the Miniserver by the so called "Loxone Link".

There are also other ways to extend the functionality, but this thesis focusses only on Loxone Link. Figure 1 shows all available Extensions and their interconnection.

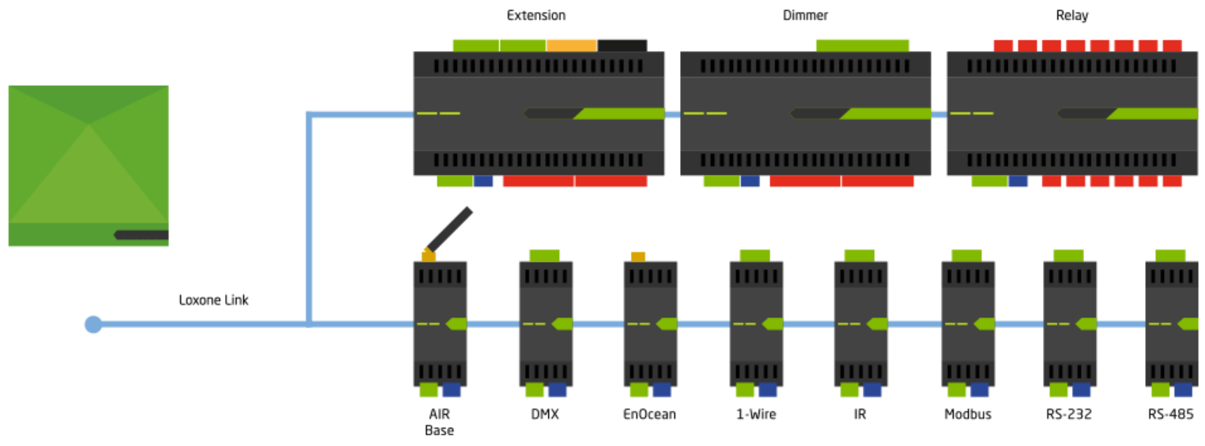


Figure 1: Loxone Smart Home Topology, Source: Loxone Shop [2]

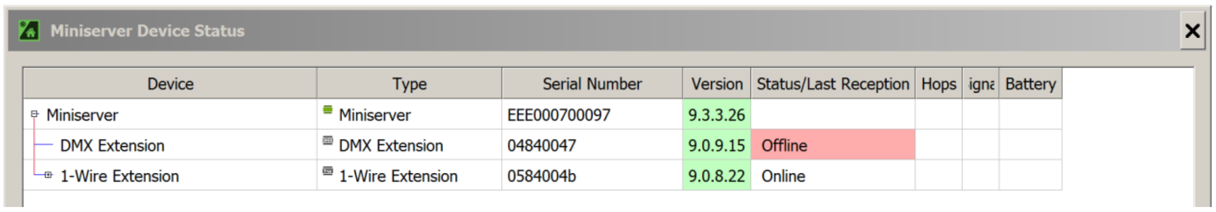
1.2.2 Loxone Config Software

A typical user interacts with his Loxone Smart Home through many ways like the smartphone app, common light-switches or a web-interface.
But when setting up a Smart Home installation the electrician has to configure a lot of things.

This is done by using a Windows Desktop Application named Loxone Config.
Here you can add Extensions to the project and connect their inputs and outputs with logic functions or scripts to get the desired behavior. This so called “configuration” is then transferred to the Miniserver with a network connection where it is stored on the Miniservers SD-card. You can also simulate a program or get a live view of the inputs and outputs. [3]

The current Loxone Config Version is 9.3.3.26. There are frequent updates and a changelog is published on the Loxone website [4]
The Loxone Config version needs to match the Miniservers Software version.
Each Extension has its own Firmware with different versions available.
The Extensions can be updated through Loxone Config [3]

In Figure 2 you can see a screenshot of an exemplary device tree with one Miniserver and two Extensions. Every device has a unique serial number and a software/firmware version. The Miniservers serial number is effectively the MAC address of its network interface. [5]


Miniserver Device Status

Device	Type	Serial Number	Version	Status/Last Reception	Hops	igna	Battery
Miniserver	Miniserver	EEE000700097	9.3.3.26				
DMX Extension	DMX Extension	04840047	9.0.9.15	Offline			
1-Wire Extension	1-Wire Extension	0584004b	9.0.8.22	Online			

Figure 2: Loxone Config – Miniserver and Extension serial numbers and versions

Loxone Config allows you to set the IP address of the Miniserver and its credentials.
This IP address allows you to access the Miniservers web interface on Port 80 with a standard browser.
The Miniserver also offers a REST API. For example, the URL to control a virtual input looks like this: "http://192.168.0.150/dev/sps/io/VI1/0". The API uses http authentication.

1.2.3 The CAN Bus

The Controller area network bus was developed to allow different control units in vehicles to communicate with each other while saving copper wiring for interconnects.
It was developed in 1983 by German Robert Bosch GmbH. It is a message based, multiplexed protocol [6]

There have been several revisions of the standard but the most current on is from the International Standardization Organization and its full name is: ISO 11898-1:20015 Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling [7] It was published in 1993.

This standard is often called CAN 2.0 and has two parts. Part A works with 11bit wide identifiers and PART B allows for 29-bit identifiers. See Figure 3 for an example of a logic-analyzer capture of a can bus packet with a 29-bit identifier. The picture shows one packet in two different zoom levels.

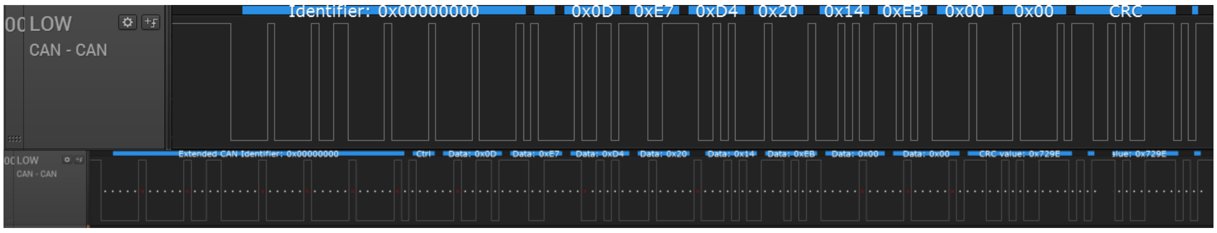


Figure 3: CAN Bus Packet Example with 29bit identifier

Bits are encoded by the two states “dominant” and “recessive”. A dominant bit is logically equivalent to zero, and a recessive bit is equivalent to a logically one.
Different nodes can transmit at the same time. Dominant bits always “win” on the bus. When a node sees its bit losing it stops transmitting and must wait for a later time to transmit again.

Therefore, the message or packet with ID 0x00000000 as seen in Figure 3 is the most important message and always “wins”. This is the only message that is always capable of fulfilling real-time requirements.

Bit rates of up to 1Mbit/s are possible for short (few meters) bus lengths. Common bitrates are 125kbit/s, 250kbit/s, 500kbit/s and 1Mbit/s. The bit rate defines the bit-timing and the nominal bit-time. To adjust for errors in phase and noise the can bus has the capability to synchronize on the bit timing. The nominal bit-time is divided in time quanta and different phases. This is used by each CAN controller to adjust its sampling frequency.

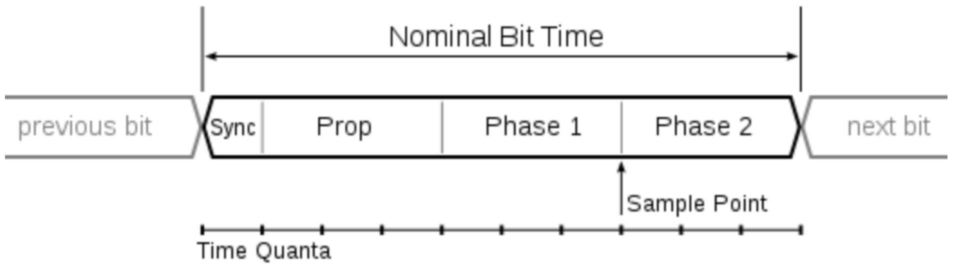


Figure 4: CAN Bus time quanta’s, Source: [8]

The physical layer specifies four lines. On a 9-pin D-sub type male connector these lines are connected as shown in Table 1: CAN-Bus Connections.

9-pin D-Sub Connector Pin	Signal name
2	CAN-Low (CAN−)
3	GND (Ground)
7	CAN-High (CAN+)
9	CAN V+ (Power)

Table 1: CAN-Bus Connections

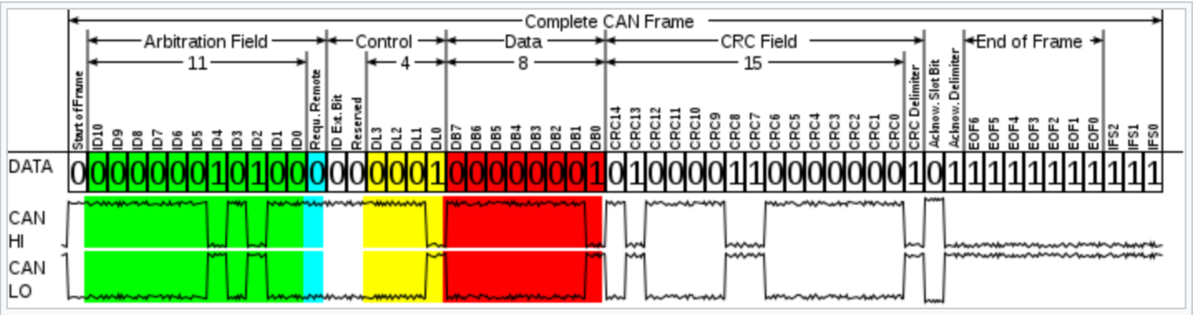
The Bus needs to be terminated with a 120 Ohm resistor on both ends to minimize reflections.

The two frame formats of CAN 2.0A and CAN 2.0B each have four frame types listed in Table 2.

Frame Type	Description
DATA Frame	CAN-Low (CAN−)
REMOTE Frame	GND (Ground)
ERROR Frame	CAN-High (CAN+)
OVERLOAD Frame	CAN V+ (Power)

Table 2: CAN-Bus Frame Types, Source: [8]

Figure 5 shows a CAN 2.0 Frame with the Signals CAN-Low and CAN-High, the corresponding logic states and the names of the fields and bits.



CAN-Frame in base format with electrical levels without stuffbits

Figure 5: CAN-Bus Data Frame

1.2.4 The DMX512 protocol

DMX512 is a standard that was originally designed to control stage lights and theatrical rigging. Since many different types of lighting dimmers exist it is also being used for lights in smart homes.
It is standardized in ANSI E1.11 – 2008, USITT DMX512-A", or just "DMX512-A [9]

A DMX-512 universe consists of a DMX controller that sends out the data and one or more daisy chained nodes. The bus is terminated with 120 Ohm on both ends.
The bitrate is 250kbit/s. Data is transmitted half duplex and asynchronous serial.

In Figure 6 you can see the beginning of a DMX transmission sequence.
It starts with the DMX line being low for at least 92 microseconds (break condition) followed by a Mark after Break and then a stop bit.
Afterwards a Start Code is transmitted. For normal lighting this is 0x00.
Then up to 512 DMX slots or channels each containing one data byte are transmitted.

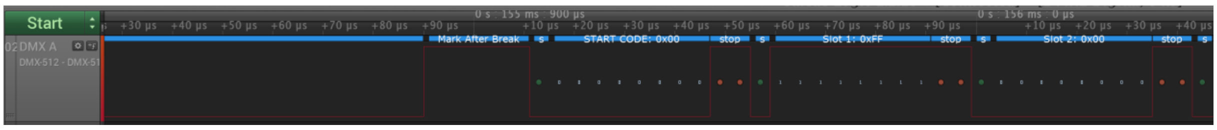


Figure 6: Beginning of a DMX transmission

1.2.5 The DMX-RDM feature

RDM is short for Remote Device Management and is a protocol enhancement of DMX-512. It is standardized in ANSI E1.20, Remote Device Management over DMX512 Networks [10]

The protocol is designed for status monitoring, remote configuration and management of DMX nodes. An example for its usage can be found in [11].

The protocol information is transmitted in normal DMX-512 slots. If a node receives a DMX packet with START Code 0xCC instead of 0x00 it knows it must interpret the DMX channels as RDM packets. The packet layout can be found in Figure 7.

Slot #	Description	Data Value	Remarks
0	START Code	0xCC	SC_RDM
1	Sub START Code	0x01	SC_SUB_MESSAGE
2	Message Length	0x19	Slot # of Checksum High = 25
3	Destination UID	0x12	0x123456789abc
4		0x34	
5		0x56	
6		0x78	
7		0x9A	
8		0xBC	
9	Source UID	0xCB	0xcba987654321
10		0xA9	
11		0x87	
12		0x65	
13		0x43	
14		0x21	
15	Transaction Number	0x00	
16	Port ID / Response Type	0x01	
17	Message Count	0x00	
18	Sub-Device	0x00	Root Device
19		0x00	
20	Command Class	0x20	GET_COMMAND
21	Parameter ID	0x00	STATUS_MESSAGES
22		0x30	
23	Parameter Data Length	0x01	
24	Parameter Data	0x04	STATUS_ERROR
25	Checksum High	0x06	0x066A
26	Checksum Low	0x6A	

Figure 7: Example of an RDM Packet and its layout, Source: [10]

1.3 State of the Art

Loxone Smart Home Hardware is relatively uncommon because of the huge global market and still comparatively new. Therefore, there is not a lot of information about it in “classical” scientific journals or papers. Some research into Loxone IT security has been done by Austrian Company SEC Consult. [12] [13]

There are a few quite active online forums that discuss the Loxone Link.
Both loxwiki.eu [14]. and mikcrocontroller.net [15] have an article about Loxone Link, but the mikcrocontroller.net article is much more detailed. It is based on a forum discussion that takes place in the thread named “Loxone Link - CAN-Bus”. [16]
There is another forum thread on loxforum.com [17] that has some pictures of Loxone Link related hardware.

The next two Chapters are all based on the above-mentioned sources and should give you an overview of the state of the art in reverse engineering of the Loxone Link protocol.

1.3.1 Loxone Link CAN Bus Parameters

Loxone Link is based on the CAN Bus and uses a bit-rate of 125kbit/s.
The CAN DLC is always 8 bytes. There is a direct relationship between the serial number of an Extension and the CAN ID it uses to transmit and receive messages.
When an Extension sends data to the Miniserver it uses an ID that is equal to 0x0abcdefg and when the Miniserver sends data to the Extension it uses the ID 0x1abcdefg. The nibbles “abcdefg” are the serial number of the extension. Nibble “a” encodes the type of extension. See Table 3 for a list of Extensions and their serial numbers

Extension serial number	Extension type
0x01xxxxxx	Extention [sic]
0x02xxxxxx	Dimmer Extention[sic]
0x03xxxxxx	EnOcean
0x04xxxxxx	DMX
0x05xxxxxx	1-Wire
0x06xxxxxx	RS232
0x07xxxxxx	RS485
0x08xxxxxx	IR
0x09xxxxxx	Modbus
0x0Bxxxxxx	Relais
0x0C00001	Air Base (Miniserver Go)
0x0Cxxxxxx	Air Base
0x0Dxxxxxx	DALI
0x0Fxxxxxx	Fröhling
0x12xxxxxx	Internorm
0x13xxxxxx	Tree
0x14xxxxxx	Digital Input (DI)
0x01xxxxxx	Extention[sic]
0x02xxxxxx	Dimmer Extention[sic]
0x03xxxxxx	EnOcean

Table 3: Relationship between Extension serial number and Extension type, Source: [15]

1.3.2 Loxone Link CAN Packets and their Structure

To find Extensions on the bus the Miniserver sends the packet seen in Figure 8.

Time[s]	#[d]	ID[x]	Data[x]								CRC[x]	ACK
			0	1	2	3	4	5	6	7		
0.0000	0	00000000	0B	00	00	00	6E	95	6B	00	????	ACK

Figure 8: Extension find packet, Source: [15]

To identify an Extension with serial number “abcdefg” in the switch cabinet by making its LED blink the Miniserver sends the packet seen in Figure 9: Extension identify packet, Source: [15].

Time[s]	#[d]	ID[x]	Data[x]								CRC[x]	ACK
			0	1	2	3	4	5	6	7		
0.0000	0	0abcdefg	08	3B	00	00	00	00	00	00	????	ACK

Figure 9: Extension identify packet, Source: [15]

The packet layout of Alive and Online Messages can be seen in Figure 10: Alive and Online Messages, Source: [15]
Packet 0 shows an Extension sending an Alive message to the Miniserver using its own serial number as ID. Byte 4 to 7 encode the Extensions firmware version.
Packet 1 is an Alive Message from the Miniserver to the Extension.
Packet 2 is an Online Message from an Extension to the Miniserver using its own serial number as ID. Byte 4 to 7 encode the Extensions firmware version.

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	xxxxxxxx	89	00	00	00	6E	95	6B	00	????	ACK
1	1xxxxxxx	0F	56	8F	21	B0	56	8F	21	????	
2	xxxxxxxx	8F	00	00	00	6E	95	6B	00		

Figure 10: Alive and Online Messages, Source: [15]

1.4 Problem / Research Question

This thesis aims to identify the meaning and function of every bit on a proprietary protocol called Loxone Link that is implemented on top of a CAN Bus. To limit the scope of the thesis to a sensible extent, this thesis will only focus on the communication between a Miniserver and one DMX Extension.

2 Main Body

This chapter shows the contribution and extension this thesis provides based on the state of the art as described in Section 1.3.

2.1 Background

Full Disclosure: In 2012 and 2013 I worked at Loxone Electronics as Embedded Developer. This is where I gained my first experiences with the topic of smart homes.
However please note that none of my findings are based on source code or other information that was available to me during my time of employment.

2.2 Solution Approach

This thesis uses an empirical method in the form of Loxone test-installation like the one in Figure 11. Carefully selected stimuli are inserted in the black box system and the results are measured with appropriate hardware and software. The data is compared to similar data from other work and the findings are then documented.

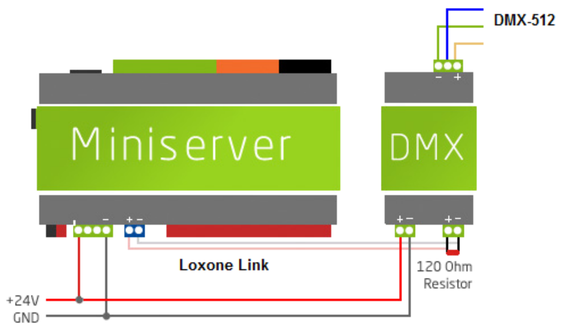


Figure 11: Miniserver connected to DMX Extension, Source: Loxone Documentation [18], modified.

For all tests Loxone Config in Version 9.3.3.26 was used on a Windows 7 Professional SP1 desktop computer.
The Miniserver used ran on software version 9.3.3.26 and its serial number is EEE000700097
The DMX Extension ran on firmware version 9.0.9.15 and its serial number is 04840047.
The 1-WIRE Extension ran on firmware version 9.0.8.22 and its serial number is 0584004B.

Most external stimuli to the system were created by carefully crafted Loxone Config Files and triggered by the REST API. The API for accessing I/O lines is called by a HTTP POST to the API URL at “admin:admin@192.168.0.150/dev/sps/io/”.

CAN Bus traffic was captured with a “Saleae Logic 8” logic analyzer and decoded with the matching software [19]. The logic analyzers 0-5V TTL inputs where matched to can bus voltage levels by using a can bus transceiver. An example of a capture screenshot can be seen in Figure 3 in Section 1.2.3.

Captured CAN Bus packets are visualized like in Figure 12.
One row corresponds to one CAN Bus Frame. Only the columns “ID”, “Data”, “CRC” and “ACK” are actual CAN Bus data. The rest is metadata specific to the test case.

Time[s]	#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	SUB[d]
		0 1 2	0	1	2	3	4	5	6	7	0 1			
2.0869	0	14840047	44	00	0D	00	08	00	BD	01	6176	ACK	1	

Figure 12: Example of a visualization of a can bus capture.

DMX traffic was also level converted by using a transceiver IC as seen in Figure 13. The receiver output “RO” was connected to one logic analyzer channel.

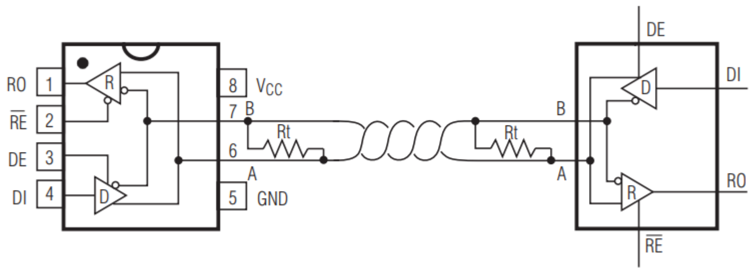


Figure 13: DMX Transceiver

The following Chapters show in detail how each test case was set up and what knowledge could be gained from the captured bus data.

To better understand the CAN packets please remember what we already know from the State of the Art as described in Chapter 1.3.2 Loxone Link CAN Packets and their Structure
Packets with ID 0x106FF010 and 0x00000000 are from the Miniserver.
Packets with ID 0x04840047 and 0x14840047 belong to an Extension with serial number 4840047.

2.2.1 DMX Extension - DMX Address and Data

The Loxone DMX Extension supports six different types of DMX Peripherals. These are listed in Table 4.

Original Name in Loxone Config	Name for the purposes of this thesis
DMX Actuator	DMX 1 Actuator
DMX 3 Actuator	DMX 3 Actuator
DMX RGB Actuator	DMX RGB Actuator
DMX 4 Actuator	DMX 4 Actuator
DMX RGB and W Actuator	DMX RGBW Actuator
DMX actuator Lumitech	DMX Lumitech Actuator

Table 4: Supported Types of DMX Peripherals

For each type of peripheral a separate Miniserver configuration was created and then used to input specific stimuli into the system. Then the response was measured by logging can bus traffic and DMX traffic. In Figure 14 you can see a capture of the DMX data stream.
Data output of all 512 DMX channels takes ~23ms followed by a ~10ms break.

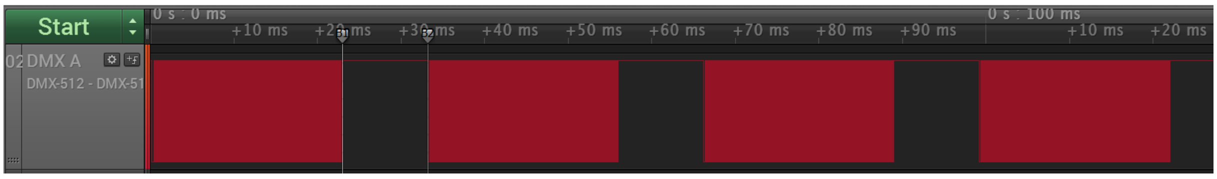


Figure 14: DMX Output by DMX Extension

Figure 15 shows a closeup of the start of a DMX output sequence including the start code (0x00 for lighting data) and channel 1 data (0xFF).

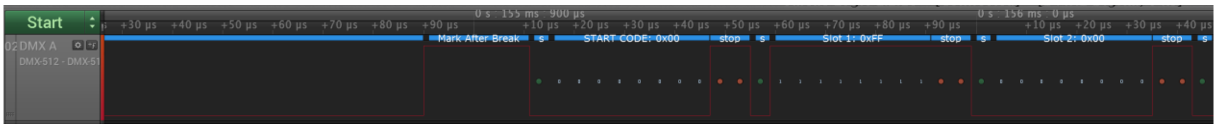


Figure 15: The first DMX channel

2.2.1.1 DMX 1 Actuator Address Sweep

First a Miniserver Configuration as seen in Figure 16 was used, which allowed to send a DMX value change to 5 separate DMX 1 Actuators.
These Actuators where configured with DMX Addresses 1,2,3,4 and 512.

The new Value was set to “1” by sending a HTTP GET to “192.168.0.150/dev/sps/io/VIx/1”, where x represents the number of the respective virtual input (1,2,3,4,5).

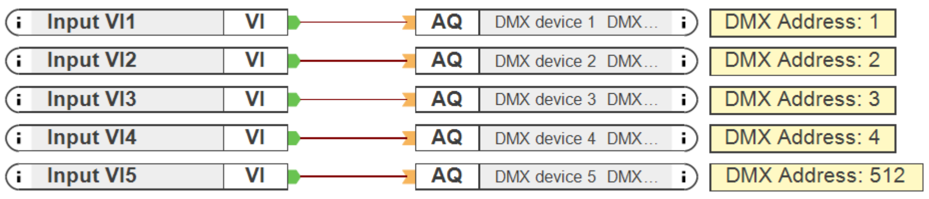


Figure 16: MS Configuration for DMX Channel sweep

Each time after setting the new Value a sequence of three packets where observed on the Loxone Link.
The original capture data can be found in Appendix A, but a subset of the packets is shown in Figure 17. DMX Values are shown in column “RAW[d]”, DMX channels are shown in column “CH[d]” and the column “SEQ[d]” indicates the sequence number to distinguish between the three packets that make up one value change command. Packet 0,1 and 2 are part of the value change command to DMX Address 1 and Packet 3,4 and 5 are part of the value change command to DMX Address 2.

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
0	14840047	44	00	0D	00	08	00	BD	01	6176	ACK	1	1	1
1	14840047	44	01	00	E4	00	00	02	37	267C	ACK	1	1	2
2	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	1	3
3	14840047	44	00	0D	00	08	00	BE	01	6885	ACK	1	2	1
4	14840047	44	01	00	E4	01	00	02	37	3884	ACK	1	2	2
5	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	2	3

Figure 17: Loxone Link packet capture during DMX channel sweep

To clearly see the changes between each channel we compare all packets with the same SEQ numbers. Figure 18 shows all packets with SEQ=1, Figure 19 shows all packets with SEQ=2 and Figure 20 shows all packets with SEQ=3.

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
0	14840047	44	00	0D	00	08	00	BD	01	6176	ACK	1	1	1
3	14840047	44	00	0D	00	08	00	BE	01	6885	ACK	1	2	1
6	14840047	44	00	0D	00	08	00	BF	01	2CA3	ACK	1	3	1
9	14840047	44	00	0D	00	08	00	C0	01	4430	ACK	1	4	1
12	14840047	44	00	0D	00	08	00	BD	02	6A44	ACK	1	512	1

Figure 18: Packets with SEQ=1

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
1	14840047	44	01	00	E4	00	00	02	37	267C	ACK	1	1	2
4	14840047	44	01	00	E4	01	00	02	37	3884	ACK	1	2	2
7	14840047	44	01	00	E4	02	00	02	37	1B8C	ACK	1	3	2
10	14840047	44	01	00	E4	03	00	02	37	0574	ACK	1	4	2
13	14840047	44	01	00	E4	FF	01	02	37	7E98	ACK	1	512	2

Figure 19: Packets with SEQ=2

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
2	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	1	3
5	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	2	3
8	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	3	3
11	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	4	3
14	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	512	3

Figure 20: Packets with SEQ=3

From this comparison we gain a lot of insights:

- SEQ=1 packets: Byte 6 and Byte 7 change with each packet.
- SEQ=1 packets: Bytes 0,2,3,4,5 are constant.
- SEQ=2 packets: Byte 4 and Byte 5 encode the DMX channel number.
- SEQ=2 packets: Bytes 0,2,3,6,7 are constant.
- SEQ=3 Packets: All Bytes are constant.
- Byte 1 of each packet encodes its sequence number starting with 0x00.
- All of the packets start with 0x44.

2.2.1.2 DMX 1 Actuator Value Sweep

Here a Miniserver Configuration as seen in Figure 21 was used, which allowed to send a DMX value change to a single DMX 1 Actuator which was configured with DMX Addresses 1.

The new value was set by sending a HTTP GET to “192.168.0.150/dev/sps/io/VI1/x”, where x represents the new value (0,1,2,3,4,5,100).

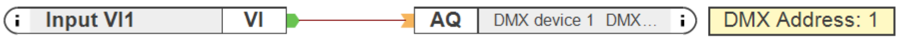


Figure 21: MS Configuration for DMX Value sweep

As we already know from chapter 2.2.1.1, setting a new value results in a sequence of three packets on the Loxone Link.
The original capture data can be found in Appendix B and a subset is shown in Figure 22.

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
0	14840047	44	00	0D	00	08	00	BB	01	7290	ACK	0	1	1
1	14840047	44	01	00	E4	00	00	00	37	6BA9	ACK	0	1	2
2	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	0	1	3
18	14840047	44	00	0D	00	08	00	BA	02	3D84	ACK	100	1	1
19	14840047	44	01	00	E4	00	00	FF	37	3B30	ACK	100	1	2
20	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	100	1	3

Figure 22: Loxone Link packet capture during DMX value sweep

Once again, we use the method of comparing groups of packets with the same SEQ number. Packets with SEQ=1 and SEQ=3 show the same behavior as seen in Chapter 2.2.1.1. Packets with SEQ=2 however show some changes relating to different set values. They are highlighted in Figure 23.

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
1	14840047	44	01	00	E4	00	00	00	37	6BA9	ACK	0	1	2
4	14840047	44	01	00	E4	00	00	02	37	267C	ACK	1	1	2
7	14840047	44	01	00	E4	00	00	05	37	71BC	ACK	2	1	2
10	14840047	44	01	00	E4	00	00	07	37	3C69	ACK	3	1	2
13	14840047	44	01	00	E4	00	00	0A	37	5F83	ACK	4	1	2
16	14840047	44	01	00	E4	00	00	0C	37	4C65	ACK	5	1	2
19	14840047	44	01	00	E4	00	00	FF	37	3B30	ACK	100	1	2

Figure 23: Packets with SEQ=2

This suggests that Byte 6 of every SEQ=2 packet encodes the DMX set value.

2.2.1.3 DMX 3 Actuator Value Sweep

Captured Loxone Link Data seems to suggest that a DMX3 Actuator works exactly like three separate DMX 1 Actuators. If you set up a DMX 3 Actuator with Address 1 it behaves like three DMX 1 Actuators with Addresses 1,2 and 3.

2.2.1.4 DMX RGB Actuator Value Sweep

A Miniserver Configuration as seen in Figure 24 was used, which allowed to send a DMX value change to a single DMX RGB Actuator, which was configured with DMX Address 1.

The new value was set by sending a HTTP GET to “192.168.0.150/dev/sps/io/VI1/x”, where x represents the new value (0,100,100.000,100.000.000). Those values where chosen because Loxone encodes the three color-components in a way that a value of 100 means 100% Green, 100.000 means 100% Red and 100.000.000 means 100% Blue.

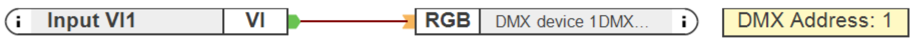


Figure 24: MS Configuration for DMX Value sweep

We do yet another comparison of groups of packets with the same SEQ number and highlight the changes. You can see the results in Figure 25 to Figure 27.

The full packet capture can be found in Appendix C.

#[d]	ID[x]	Data[x] 0	1	2	3	4	5	6	7	CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
0	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	0	1	1
3	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	100	1	1
6	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	10^5	1	1
9	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	10^8	1	1

Figure 25: Packets with SEQ=1

#[d]	ID[x]	Data[x] 0	1	2	3	4	5	6	7	CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
1	14840047	44	01	**01**	E4	00	00	**00**	**00**	50C1	ACK	0	1	2
4	14840047	44	01	**01**	E4	00	00	**FF**	**00**	0058	ACK	100	1	2
7	14840047	44	01	**01**	E4	00	00	**00**	**FF**	5054	ACK	10^5	1	2
10	14840047	44	01	**01**	E4	00	00	**00**	**00**	50C1	ACK	10^8	1	2

Figure 26: Packets with SEQ=2

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
2	14840047	44	02	**00**	00	30	E7	D4	20	0601	ACK	0	1	3
5	14840047	44	02	**00**	00	30	E7	D4	20	0601	ACK	100	1	3
8	14840047	44	02	**00**	00	30	E7	D4	20	0601	ACK	10^5	1	3
11	14840047	44	02	**FF**	00	30	E7	D4	20	62BD	ACK	10^8	1	3

Figure 27: Packets with SEQ=1

By making this comparison the following knowledge is gained:

- SEQ=1 Packets: All Bytes are constant.
- SEQ=2 packets: Byte 6 encodes the RED value, Byte 7 encodes the GREEN value.
- SEQ=2 packets: Byte 2 encodes the type of actuator (compare to Figure 19 Byte 2)
- SEQ=3 packets: Bytes 2 encodes the BLUE value.
- The rest of the Bytes are assumed to be constant for now.

2.2.1.5 DMX 4 Actuator Value Sweep

Just like the DMX 3 Actuator in 2.2.1.3 the DMX 4 Actuator is a simple combination of four separate DMX 1 Actuators.

2.2.1.6 DMX RGBW Actuator Value Sweep

The RGBW Actuator is a combination of an RGB Actuator and a single DMX 1 Actuator for the W(White) Channel.

2.2.1.7 DMX Lumitech Actuator Value Sweep

A Miniserver Configuration as seen in Figure 28 was used to test the behavior of a DMX Lumitech Actuator. Every time the virtual Input VI1 is triggered by a HTTP GET the Lighting Controller V1 Block sends the next “Scene” to the DMX Lumitech Actuator.

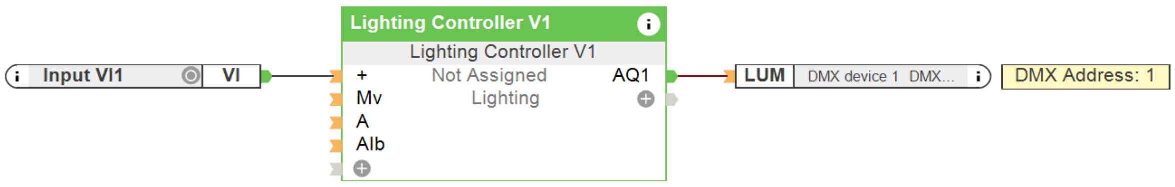
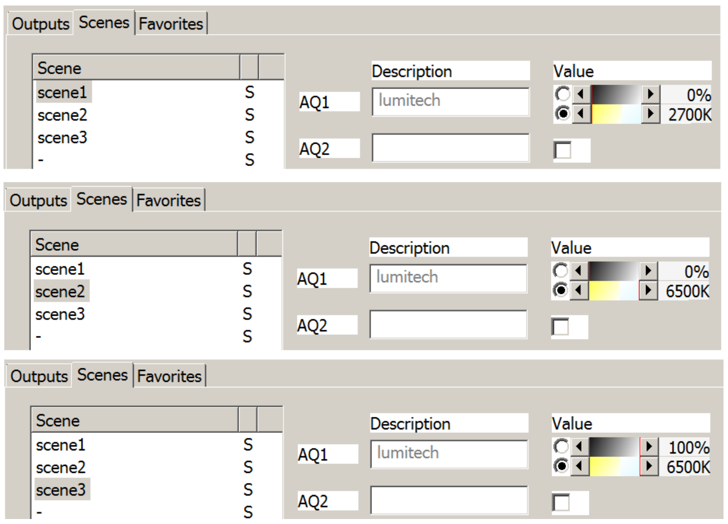


Figure 28: MS Configuration for DMX Value sweep

A scene is defined by two values - brightness and color temperature.
Three scenes where configured in the Lighting Controller V1 block as seen in Figure 29.
Scene 1: 0% / 2700K, Scene 2: 0% / 6500K, Scene 3: 100% / 6500K



Scene	
scene1	S
scene2	S
scene3	S
-	S

	Description	Value
AQ1	lumitech	0% / 2700K
AQ2		☐

Scene	
scene1	S
scene2	S
scene3	S
-	S

	Description	Value
AQ1	lumitech	0% / 6500K
AQ2		☐

Scene	
scene1	S
scene2	S
scene3	S
-	S

	Description	Value
AQ1	lumitech	100% / 6500K
AQ2		☐

Figure 29: Scene configuration of the Lightning Controller V1 block

The groups of packets with the same SEQ number are compared with each other and the changes are highlighted.

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	SEQ[d]
		0	1	2	3	4	5	6	7				
0	14840047	44	00	0D	00	08	00	07	01	37FF	ACK	0%/2700K	1
3	14840047	44	00	0D	00	08	00	06	02	78EB	ACK	0%/6500K	1
6	14840047	44	00	0D	00	08	00	E5	02	39D9	ACK	100%/6500K	1
9	14840047	44	00	0D	00	08	00	85	02	4912	ACK	100%/4600K	1

Figure 30: Packets with SEQ=1

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	SEQ[d]
		0	1	2	3	4	5	6	7				
1	14840047	44	01	03	E4	00	00	00	00	6C99	ACK	0%/2700K	2
4	14840047	44	01	03	E4	00	00	00	00	6C99	ACK	0%/6500K	2
7	14840047	44	01	03	E4	00	00	00	FF	6C0C	ACK	100%/6500K	2
10	14840047	44	01	03	E4	00	00	00	FF	6C0C	ACK	100%/4600K	2

Figure 31: Packets with SEQ=2

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	SEQ[d]
		0	1	2	3	4	5	6	7				
2	14840047	44	02	00	20	30	E7	D4	20	6708	ACK	0%/2700K	3
5	14840047	44	02	FF	20	30	E7	D4	20	03B4	ACK	0%/6500K	3
8	14840047	44	02	FF	00	00	00	00	00	085D	ACK	100%/6500K	3
11	14840047	44	02	7F	20	30	E7	D4	20	5A40	ACK	100%/4600K	3

Figure 32: Packets with SEQ=3

With this comparison we gain the following knowledge:

- SEQ=1 packets: Byte 6 and Byte 7 change with RAW values.
- SEQ=2 packets: Byte 7 encodes the brightness value.
- SEQ=2 packets: Byte 2 encodes the type of actuator (compare to Figure 19 Byte 2)
- SEQ=3 packets: Byte 2 encodes the color temperature.
- The rest of the Bytes are assumed to be constant for now.

2.2.2 DMX Extension - IDENTIFY

Loxone Config has a function that makes it easy to identify a specific extension in the switch cabinet by making the LED of this extension blink.
Figure 33 and Figure 34 show the bus traffic on the Loxone Link while the Extension is selected for identification. In addition to the DMX Extension a 1-WIRE Extension has been used to identify any differences in the protocol.

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
1	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
2	106FF010	FF	00	08	07	**47**	00	84	**04**	4B39	ACK
3	**14840047**	08	08	00	00	00	00	00	00	7B49	ACK
4	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
5	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
6	**04840047**	8D	00	00	00	**D3**	**57**	**89**	**00**	63B0	ACK
7	00000000	0D	E7	D4	20	**E8**	**17**	00	00	0967	ACK
8	00000000	2D	E2	07	1B	**CD**	**96**	**1D**	03	7A5F	ACK
9	106FF00C	FF	E2	07	1B	**CD**	**96**	**1D**	03	0936	ACK

Figure 33: DMX Extension during IDENTIFY

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
1	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
2	106FF010	FF	00	08	07	**4B**	00	84	**05**	4719	ACK
3	**1584004B**	08	08	00	00	00	00	00	00	53AA	ACK
4	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
5	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
6	**0584004B**	8D	00	00	00	**76**	**57**	**89**	**00**	1142	ACK
7	00000000	0D	E7	D4	20	**78**	**87**	00	00	327E	ACK
8	00000000	2D	E2	07	1B	**9A**	**B9**	**1A**	03	467A	ACK
9	106FF00C	FF	E2	07	1B	**9A**	**B9**	**1A**	03	3513	ACK

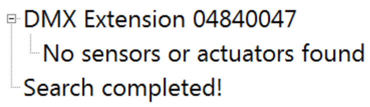
Figure 34: 1-WIRE Extension during IDENTIFY

By making this comparison we gain following knowledge:

- Packet 0 and 1 are constant and probably initiate the identification procedure.
- Packet 3 is from the Miniserver to the Extension and starts the blinking of the LED.
- Packet 4 and 5 are constant
- Packet 6 originates from the Extension. Byte 4 - 7 encode the Firmware Version
- The rest of the packets are mostly constant with some varying bytes whose purpose could not be determined.

2.2.3 DMX Extension - RDM DISCOVERY

To trigger a DMX RDM Discovery Request on the DMX Line, a DMX Device Search was started from Loxone Config. The result can be seen in Figure 35. Unfortunately, no RDM capable DMX hardware was available for probing, so we can only discuss the consequences of a discovery request.



DMX Extension 04840047
No sensors or actuators found
Search completed!

Figure 35: Status Output from Loxone Config

Figure 36 shows the packets observed on the Loxone Link during an RDM device search. Packet 0 is sent from the Miniserver to the Extension as indicated by the CAN ID. Its purpose is to command the DMX Extension to start the device search.
Packet 1 is sent from the DMX Extension to the Miniserver and tells it that the device search has finished. The process takes ~357ms.

Time [s]	#[d]	ID[x]	Data[x]								CRC[x]	ACK
			0	1	2	3	4	5	6	7		
3.424777	0	14840047	64	B7	00	FF	00	B5	1B	20	18D2	ACK
3.781665	1	4840047	E5	00	00	00	00	00	00	00	6B81	ACK

Figure 36: RDM Discovery Request

Figure 37 shows the DMX bus transitioning from outputting 512 channels of Lighting Data to outputting the data stream required for an RDM search. The transition happens just after the 0.4 second mark when the first Loxone Link packet is sent. The DMX Extension then resumes with outputting Lighting Data after finishing the RDM Discovery just before the 0.8 second mark.

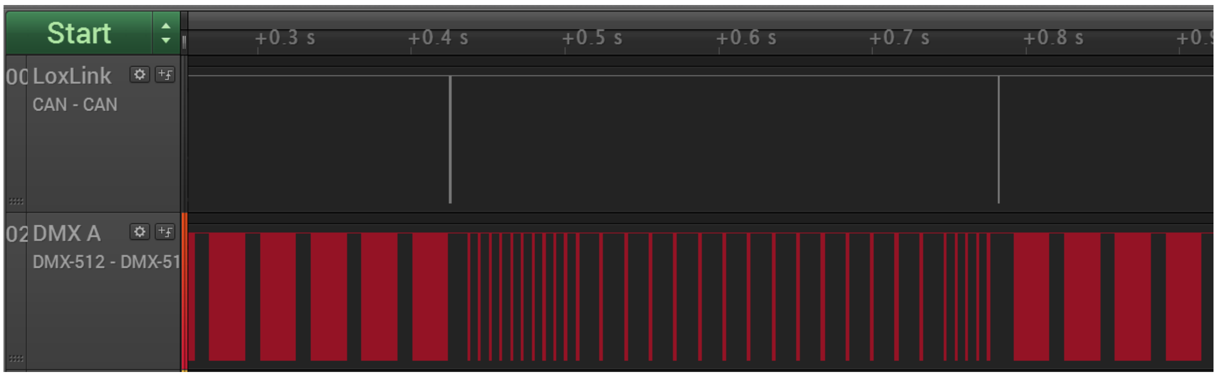
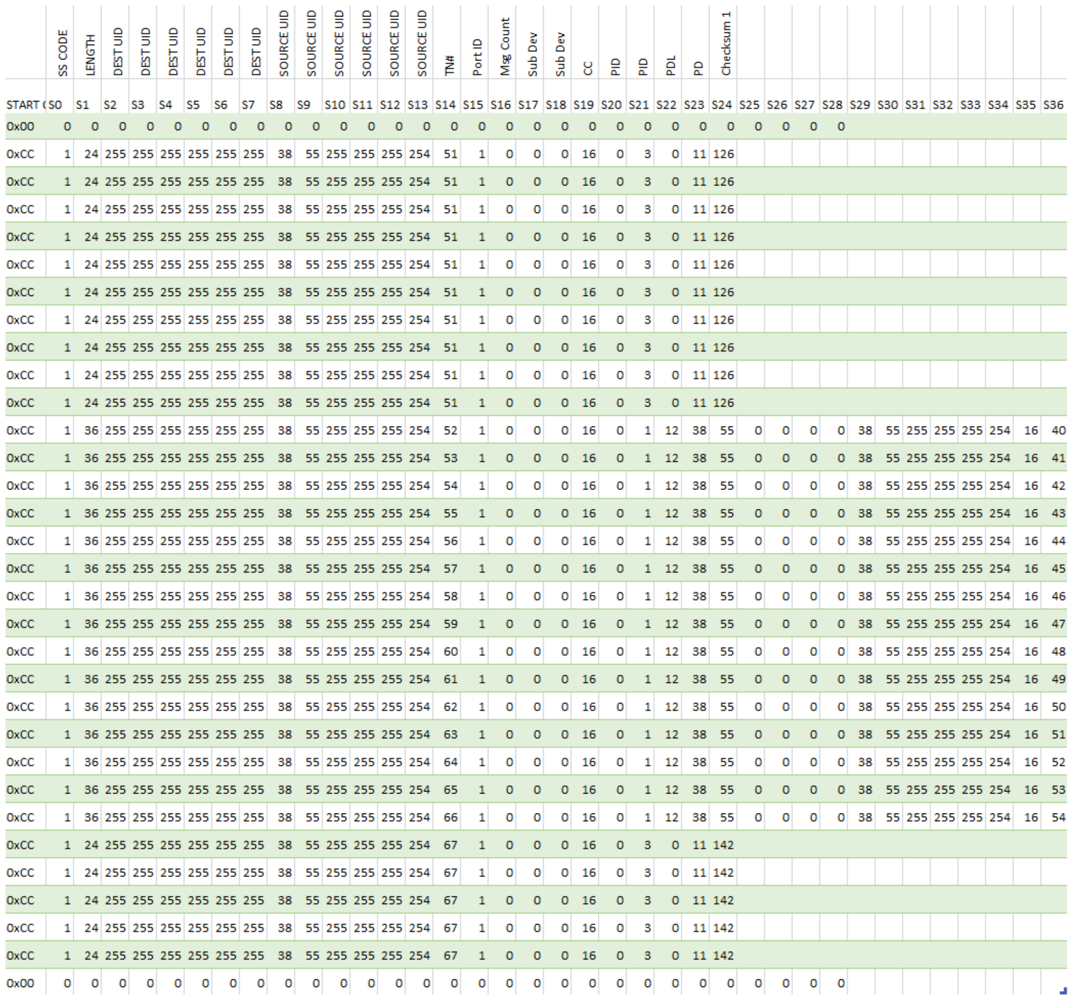


Figure 37: DMX Bus Traffic during RDM Discovery

The DMX Bus Traffic during the RDM Discovery Phase can be seen in Figure 38



	SS CODE	LENGTH	DEST UID	DEST UID	DEST UID	DEST UID	DEST UID	DEST UID	SOURCE UID	SOURCE UID	SOURCE UID	SOURCE UID	SOURCE UID	SOURCE UID	TN#	Port ID	Msg Count	Sub Dev	Sub Dev	CC	PID	PID	PDL	PD	Checksum 1												
START (	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21	S22	S23	S24	S25	S26	S27	S28	S29	S30	S31	S32	S33	S34	S35	S36
0x00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	51	1	0	0	0	16	0	3	0	11	126												
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	52	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	40
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	53	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	41
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	54	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	42
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	55	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	43
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	56	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	44
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	57	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	45
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	58	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	46
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	59	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	47
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	60	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	48
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	61	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	49
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	62	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	50
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	63	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	51
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	64	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	52
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	65	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	53
0xCC	1	36	255	255	255	255	255	255	38	55	255	255	255	254	66	1	0	0	0	16	0	1	12	38	55	0	0	0	0	38	55	255	255	255	254	16	54
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	67	1	0	0	0	16	0	3	0	11	142												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	67	1	0	0	0	16	0	3	0	11	142												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	67	1	0	0	0	16	0	3	0	11	142												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	67	1	0	0	0	16	0	3	0	11	142												
0xCC	1	24	255	255	255	255	255	255	38	55	255	255	255	254	67	1	0	0	0	16	0	3	0	11	142												
0x00	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								

Figure 38: DMX Bus Traffic during RDM Discovery

2.2.4 Loxone Link during STARTUP

For this test both the DMX Extension and the 1-WIRE Extension where connected to the Loxone Link and the system was booted.
Two Extension have been used because with only one Extension a corner-case would occur: After enabling power, the Extension would boot quite fast (several ms) while the Miniserver takes more than 20 seconds to boot. During this time the Extension is the only node on the CAN bus and therefore no packets are being acknowledged which leads to the Extension flooding the bus.

The beginning of the startup sequence is show in Figure 39. The full capture is made available in Appendix G.

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	0x0584004B	87	00	00	00	**76**	**57**	**89**	**00**	5E6D	ACK
1	0x04840047	87	00	00	00	**D3**	**57**	**89**	**00**	2C9F	ACK
2	0x0584004B	B6	00	01	00	7C	A7	F1	63	7301	ACK
3	0x04840047	F8	00	00	00	00	00	00	00	43A0	ACK
4	0x0584004B	87	00	**01**	00	76	57	89	00	4041	ACK
5	0x04840047	87	00	**00**	00	D3	57	89	00	2C9F	ACK
⋮	⋮				⋮					⋮	⋮
56	0x0584004B	87	00	**01**	00	76	57	89	00	4041	ACK
57	0x04840047	87	00	**00**	00	D3	57	89	00	2C9F	ACK
58	0x106FF007	FF	00	00	00	00	00	00	00	3C7F	ACK

Figure 39: Loxone Link Startup Sequence

The results are:

- Packet 0 is an Alive Message from the 1-WIRE Extension with its Firmware version
- Packet 1 is an Alive Message from the DMX Extension with its Firmware version
- Packet 2 and 3 are probably some sort of status message.
- Packet 4 and 5 are again Alive Messages from the two Extensions but now with Byte 2 encoding the sequence number of the Extension as defined in Loxone Config.
- Then these two Alive Packets repeat (Packet 6-57) until the Miniserver has finished booting.
- Packet 58 is the first Packet from the Miniserver after about 30 seconds from initial power on.

2.2.5 Loxone Link during NO LOAD

For this test Loxone Link packets where captured during a time where there were no external stimuli to the system. The full capture of around 200 seconds is available in Appendix H.

Figure 40 shows some interesting parts of this traffic.

Time	#[d]	ID[x]	Data[x]								CRC[x]	ACK	SEQ[d]
			0	1	2	3	4	5	6	7			
12.01	0	04840047	89	00	00	00	**D3**	**57**	**89**	**00**	71DB	ACK	
12.02	1	14840047	0F	84	14	21	90	84	14	21	3EE8	ACK	
19.52	3	00000000	2D	E2	87	FA	**BD**	**09**	**BD**	**03**	2963	ACK	2
79.74	6	00000000	2D	E2	87	FA	**D8**	**F4**	**BD**	**03**	5969	ACK	2
139.95	9	00000000	2D	E2	87	FA	**F2**	**DF**	**BE**	**03**	56E9	ACK	2
199.14	12	00000000	2D	E2	87	FA	**0F**	**C7**	**BF**	**03**	37B1	ACK	2

Figure 40: Loxone Link during no load

We can conclude that:

- Packet 1 is an Alive Message from the DMX Extension with Firmware Number
- Packet 2 is probably an acknowledgement for this Alive Message.
- Packet 3, 6, 9 and 12 have constant parts. Byte 4 – 7 can be interpreted as a number that is counting up. The difference in value between each packet is around 60.186 which correlates to the time difference between packets that is around 60.000ms

2.3 Results and Discussion

This section summarizes the most important findings of chapters 2.2.1 to 2.2.5 and provides an overview of what is now know about the Loxone Link.

2.3.1 Writing a new value to a DMX Actuator

Figure 41 shows the packet layout of the three-packet sequence that makes up a DMX Actuator write value command. Table 5 lists all bytes and their meanings.

#[d]	ID[x]	Data[x]							
		0	1	2	3	4	5	6	7
0	MS2EXTID	**44**	SEQ#	**0D**	**00**	ADDL	ADDH	**?**	**?**
1	MS2EXTID	**44**	SEQ#	ACTT	**E4**	ADDL	ADDH	DAT1	DAT2
2	MS2EXTID	**44**	SEQ#	DAT3	**20**	ADDL	ADDH	**D4**	**20**

Figure 41: DMX Actuator write value command packet layout

byte name	meaning	
MS2EXTID	= 0x100000000 && Extension serial number	
SEQ#	sequence number: 0,1,2	
ACTT	Actuator Type:	
	Actuator	**meaning**
	DMX-1	0x00*
	RGB	0x01
	LUMITECH	0x03
ADDL	Low Byte (LSB) of the DMX Address	
ADDH	High Byte (MSB) of the DMX Address	
DAT1	**Actuator**	**meaning**
	DMX-1	DMX channel value*
	RGB	RED channel value*
	LUMITECH	0x00
DAT2	**Actuator**	**meaning**
	DMX-1	0x37
	RGB	GREEN channel value*
	LUMITECH	Brightness value*
DAT3	**Actuator**	**meaning**
	DMX-1	0x80
	RGB	BLUE channel value*
	LUMITECH	Color temp., scaling = (value-2700)/(((3800)/255))

Table 5: DMX Actuator write value command byte meaning

* The formula for scaling a 0-100% value to a 0-0xFF value is: floor(value*255/100).

2.3.2 Starting a DMX RDM Discovery

Figure 42 shows the packet layout of the two-packet sequence that makes up a DMX RDM Discovery request command. Table 6 lists all bytes and their meanings.

#[d]	ID[x]	Data[x]							
		0	1	2	3	4	5	6	7
0	MS2EXTID	64	B7	00	FF	00	B5	1B	20
1	EXT2MSID	E5	00	00	00	00	00	00	00

Figure 42: DMX RDM Discovery Request command packet layout

byte name	**meaning**
MS2EXTID	= 0x100000000 && Extension serial number
EXT2MSID	= Extension serial number

Table 6: DMX RDM Discovery Request command byte meaning

2.3.3 Extension alive message

Figure 43 shows the packet layout of an Extension alive message packet. Table 7 lists all bytes and their meanings.

#[d]	ID[x]	Data[x]							
		0	1	2	3	4	5	6	7
0	MS2EXTID	**89**	**00**	**00**	**00**	FWV3	FWV2	FWV1	FWV0

Figure 43: Extension alive message packet layout

byte name	meaning
MS2EXTID	= 0x100000000 && Extension serial number
FWVx	= Extension Firmware Version

Table 7: Extension alive message byte meaning

2.3.4 Miniserver Systick Timer

Figure 44 shows the packet layout of an Miniservers Systick Timer message.

#[d]	ID[x]	Data[x]							
		0	1	2	3	4	5	6	7
0	00000000	**2D**	**E2**	**87**	**FA**	S1	S2	S3	**03**

Figure 44: Miniserver Systick Timer

byte name	meaning
Sx	Systick Timer value

Table 8: Miniserver Systick Timer message byte meaning

3 The Conclusion

3.1 Résumé

This Thesis treated the system as a black box.
Carefully crafted stimuli were applied and triggered through the REST API. The systems response was then measured by capturing packets on the CAN Bus and on the DMX Bus.
The data created by this was analyzed and compared to prior work by others. Each test brought up some new knowledge and strengthen the confidence in already well understood parts of the packet layout.

3.2 Future Work

Even with only the DMX Extension being analyzed in this thesis there are still bytes whose functions and meanings are unknown. More work is needed to understand those.
Because of this it would be advisable to create even more test cases than those shown here.
One test case completely missing from this thesis is the process of an Extension firmware update.
To be able to test more cases and also more frequently would require a partially or fully automated test setup, possible with a method for input fuzzing an automated output analysis.
Furthermore, more data and the use of statistical analysis of the output could lead to good results.

There are still a lot of other Extensions using Loxone Link that need to be analyzed to fully understand all aspects of the Bus.

Loxone has since developed another protocol based on 6LoWPAN for communicating with wireless Extensions. This is called “Loxone Air”. There is also “Loxone Tree” which is based on RS485. Both would be interesting candidates for Reverse Engineering in a future thesis.

Bibliography

[1] Loxone Electronics GmbH, “Loxone Shop - Miniserver,” 2018. [Online]. Available: https://shop.loxone.com/enen/miniserver.html. [Accessed: 06-Jun-2018].

[2] Loxone Electronics GmbH, “Loxone Shop - Miniserver Go,” 2018. [Online]. Available: https://shop.loxone.com/enen/miniserver-go.html. [Accessed: 06-Jun-2016].

[3] loxwiki.eu, “Loxone Config,” 2018. [Online]. Available: https://www.loxwiki.eu/display/LOX/Loxone+Config+Software. [Accessed: 05-Jun-2018].

[4] Loxone Electronics GmbH, “Loxone Config Changelog,” 2018. [Online]. Available: https://www.loxone.com/enen/support/changelog-config/. [Accessed: 05-Jun-2018].

[5] openhab.org, “Article: Loxone Binding,” 2018. [Online]. Available: https://www.openhab.org/addons/bindings/loxone/. [Accessed: 06-Jun-2018].

[6] CAN-CiA, “History of CAN technology.” [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/. [Accessed: 06-Jun-2018].

[7] International Organization for Standardization, “ISO 11898-1:20015 Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling.” 2015.

[8] wikipedia, “CAN-bus,” 2018. [Online]. Available: https://en.wikipedia.org/wiki/CAN_bus. [Accessed: 06-Jun-2018].

[9] The ESTA Foundation, “ANSI E1.11 – 2008, USITT DMX512-A.” 2008.

[10] The ESTA Foundation, “ANSI E1.20 - 2010 Entertainment Technology RDM Remote Device Management Over DMX512 Networks,” 2010.

[11] H. N. Manh and C. Lee, “Design of Bi-directional RDM-DMX512 Converter for LED Lighting Control,” vol. 13, no. 2, pp. 106–115, 2013.

[12] G. Johannes, “SEC Consult Vulnerability Lab - Multiple vulnerabilities 20150514,” 2018.

[13] S. Daniel, “SEC Consult Vulnerability Lab - Multiple vulnerabilities 20150227,” 2018.

[14] loxwiki.eu, “Loxone Link,” 2018. [Online]. Available: https://www.loxwiki.eu/display/LOX/Loxone+Link. [Accessed: 05-Jun-2018].

[15] mikrocontroller.net, “Article: Loxone Link,” 2018. [Online]. Available: https://www.mikrocontroller.net/articles/Loxone_Link. [Accessed: 05-Jun-2018].

[16] L. Sven, M. Reinhard, and T. Josef, “Forum Thread: Loxone Link - CAN-Bus,” 2018. [Online]. Available: https://www.mikrocontroller.net/topic/430090. [Accessed: 05-Jun-2018].

[17] loxforum.com, “Forum Thread: Loxone Link Element,” 2018. [Online]. Available: https://www.loxforum.com/forum/hardware-zubehör-sensorik/19442-loxone-link-element. [Accessed: 05-Jun-2018].

[18] Loxone Electronics GmbH, “Loxone Documentation - Wirink Accessories,” 2018. [Online]. Available: https://www.loxone.com/enen/kb/wiring-accessories/. [Accessed:

06-Jun-2018].

[19] Saleae Corporation, “Saleae Logic 8 Datasheet.” [Online]. Available: http://downloads.saleae.com/specs/Logic+8+Data+Sheet.pdf. [Accessed: 06-Jun-2018].

List of Figures

List of Tables

List of Abbreviations

6LoWPAN	Internet Protocol (IPv6) and Low-power Wireless Personal Area Networks
ANSI	American National Standards Institute
CAN	Controller Area Network
CRC	cyclic redundancy check
DMX	Digital Multiplex
D-sub	D-subminiature
GmbH	Gesellschaft mit beschränkter Haftung (GER) = Limited Liabilty Corporation
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IR	Infrared
ISO	International Organization for Standardization
LED	light-emitting diode
MAC	media access control
RDM	Remote Device Management
REST API	Representational State Transfer Application Programming Interface
SD-card	Secure Digital card
TTL	Transistor–transistor logic
UID	Unique Identification
URL	Uniform Resource Locator
USITT	United States Institute for Theatre Technology

A: DMX-1 Actuator Channel Sweep

#[d]	ID[x]	Data[x] 0	1	2	3	4	5	6	7	CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
0	14840047	44	00	0D	00	08	00	BD	01	6176	ACK	1	1	1
1	14840047	44	01	00	E4	00	00	02	37	267C	ACK	1	1	2
2	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	1	3
3	14840047	44	00	0D	00	08	00	BE	01	6885	ACK	1	2	1
4	14840047	44	01	00	E4	01	00	02	37	3884	ACK	1	2	2
5	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	2	3
6	14840047	44	00	0D	00	08	00	BF	01	2CA3	ACK	1	3	1
7	14840047	44	01	00	E4	02	00	02	37	1B8C	ACK	1	3	2
8	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	3	3
9	14840047	44	00	0D	00	08	00	C0	01	4430	ACK	1	4	1
10	14840047	44	01	00	E4	03	00	02	37	0574	ACK	1	4	2
11	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	4	3
12	14840047	44	00	0D	00	08	00	BD	02	6A44	ACK	1	512	1
13	14840047	44	01	00	E4	FF	01	02	37	7E98	ACK	1	512	2
14	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	512	3

B: DMX-1 Actuator Value Sweep

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
0	14840047	44	00	0D	00	08	00	BB	01	7290	ACK	0	1	1
1	14840047	44	01	00	E4	00	00	00	37	6BA9	ACK	0	1	2
2	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	0	1	3
3	14840047	44	00	0D	00	08	00	BD	01	6176	ACK	1	1	1
4	14840047	44	01	00	E4	00	00	02	37	267C	ACK	1	1	2
5	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	1	1	3
6	14840047	44	00	0D	00	08	00	C0	01	4430	ACK	2	1	1
7	14840047	44	01	00	E4	00	00	05	37	71BC	ACK	2	1	2
8	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	2	1	3
9	14840047	44	00	0D	00	08	00	C2	01	09E5	ACK	3	1	1
10	14840047	44	01	00	E4	00	00	07	37	3C69	ACK	3	1	2
11	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	3	1	3
12	14840047	44	00	0D	00	08	00	C5	01	5E25	ACK	4	1	1
13	14840047	44	01	00	E4	00	00	0A	37	5F83	ACK	4	1	2
14	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	4	1	3
15	14840047	44	00	0D	00	08	00	C7	01	13F0	ACK	5	1	1
16	14840047	44	01	00	E4	00	00	0C	37	4C65	ACK	5	1	2
17	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	5	1	3
18	14840047	44	00	0D	00	08	00	BA	02	3D84	ACK	100	1	1
19	14840047	44	01	00	E4	00	00	FF	37	3B30	ACK	100	1	2
20	14840047	44	02	80	20	30	E7	D4	20	3EFC	ACK	100	1	3

C: DMX-RGB Actuator Value Sweep

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	CH[d]	SEQ[d]
		0	1	2	3	4	5	6	7					
0	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	0	1	1
1	14840047	44	01	01	E4	00	00	00	00	50C1	ACK	0	1	2
2	14840047	44	02	00	00	30	E7	D4	20	0601	ACK	0	1	3
3	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	100	1	1
4	14840047	44	01	01	E4	00	00	FF	00	0058	ACK	100	1	2
5	14840047	44	02	00	00	30	E7	D4	20	0601	ACK	100	1	3
6	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	10^5	1	1
7	14840047	44	01	01	E4	00	00	00	FF	5054	ACK	10^5	1	2
8	14840047	44	02	00	00	30	E7	D4	20	0601	ACK	10^5	1	3
9	14840047	44	00	0D	00	08	00	E4	01	76CD	ACK	10^8	1	1
10	14840047	44	01	01	E4	00	00	00	00	50C1	ACK	10^8	1	2
11	14840047	44	02	FF	00	30	E7	D4	20	62BD	ACK	10^8	1	3

D: DMX Lumitech Actuator Value Sweep

#[d]	ID[x]	Data[x]								CRC[x]	ACK	RAW[d]	SEQ[d]
		0	1	2	3	4	5	6	7				
0	14840047	44	00	0D	00	08	00	07	01	37FF	ACK	0%/2700K	1
1	14840047	44	01	03	E4	00	00	00	00	6C99	ACK	0%/2700K	2
2	14840047	44	02	00	20	30	E7	D4	20	6708	ACK	0%/2700K	3
3	14840047	44	00	0D	00	08	00	06	02	78EB	ACK	0%/6500K	1
4	14840047	44	01	03	E4	00	00	00	00	6C99	ACK	0%/6500K	2
5	14840047	44	02	FF	20	30	E7	D4	20	03B4	ACK	0%/6500K	3
6	14840047	44	00	0D	00	08	00	E5	02	39D9	ACK	100%/6500K	1
7	14840047	44	01	03	E4	00	00	00	FF	6C0C	ACK	100%/6500K	2
8	14840047	44	02	FF	00	00	00	00	00	085D	ACK	100%/6500K	3
9	14840047	44	00	0D	00	08	00	85	02	4912	ACK	100%/4600K	1
10	14840047	44	01	03	E4	00	00	00	FF	6C0C	ACK	100%/4600K	2
11	14840047	44	02	7F	20	30	E7	D4	20	5A40	ACK	100%/4600K	3

E: DMX Identify Blink

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
1	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
2	106FF010	FF	00	08	07	47	00	84	04	4B39	ACK
3	14840047	08	08	00	00	00	00	00	00	7B49	ACK
4	106FF010	FF	00	00	00	00	00	00	00	7917	ACK
5	00000000	08	CE	45	20	5C	C8	45	20	25EA	ACK
6	4840047	8D	00	00	00	D3	57	89	00	63B0	ACK
7	00000000	0D	E7	D4	20	E8	17	00	00	967	ACK
8	00000000	2D	E2	07	1B	CD	96	1D	03	7A5F	ACK
9	106FF00C	FF	E2	07	1B	CD	96	1D	03	0936	ACK

F: DMX RDM Discovery Request

Time [s]	#[d]	ID[x]	Data[x]								CRC[x]	ACK
			0	1	2	3	4	5	6	7		
3.424777	0	14840047	64	B7	00	FF	00	B5	1B	20	18D2	ACK
3.781665	1	4840047	E5	00	00	00	00	00	00	00	6B81	ACK

G: Loxone Link Startup Sequence

#[d]	ID[x]	Data[x]								CRC[x]	ACK
		0	1	2	3	4	5	6	7		
0	0x0584004B	87	00	00	00	76	57	89	00	5E6D	ACK
1	0x04840047	87	00	00	00	D3	57	89	00	2C9F	ACK
2	0x0584004B	B6	00	01	00	7C	A7	F1	63	7301	ACK
3	0x04840047	F8	00	00	00	00	00	00	00	43A0	ACK
4	0x0584004B	87	00	01	00	76	57	89	00	4041	ACK
5	0x04840047	87	00	00	00	D3	57	89	00	2C9F	ACK
⋮	⋮				⋮					⋮	⋮
56	0x0584004B	87	00	01	00	76	57	89	00	4041	ACK
57	0x04840047	87	00	00	00	D3	57	89	00	2C9F	ACK
58	0x106FF007	FF	00	00	00	00	00	00	00	3C7F	ACK
59	0x00000000	0C	00	00	00	00	00	00	00	0B12	ACK
60	0x106FF007	FF	00	00	00	00	00	00	00	3C7F	ACK
61	0x00000000	0C	00	00	00	00	00	00	00	0B12	ACK
62	0x14840047	00	2D	19	21	00	00	00	00	3438	ACK
63	0x04840047	87	00	00	00	D3	57	89	00	2C9F	ACK
64	0x04840047	F8	00	00	00	00	00	00	00	43A0	ACK
65	0x14840047	0E	BE	32	21	04	00	00	00	638D	ACK
66	0x1584004B	00	2D	19	21	00	00	00	00	1CDB	ACK
67	0x0584004B	87	00	01	00	76	57	89	00	4041	ACK
68	0x106FF0F4	00	00	00	00	00	00	00	00	393E	ACK
69	0x00000000	0B	00	01	00	76	57	89	00	73B0	ACK
70	0x0584004B	B6	00	01	00	7C	A7	F1	63	7301	ACK
71	0x1584004B	0E	F1	12	20	08	00	00	00	0B07	ACK
72	0x00000000	0D	E7	D4	20	00	00	00	00	6029	ACK
73	0x00000000	2D	E2	07	1B	9B	BF	FA	02	3B09	ACK

H: Loxone Link No Load Traffic

Time	#[d]	ID[x]	Data[x]								CRC[x]	ACK	SEQ[d]
			0	1	2	3	4	5	6	7			
12.01	0	04840047	89	00	00	00	D3	57	89	00	71DB	ACK	
12.02	1	14840047	0F	84	14	21	90	84	14	21	3EE8	ACK	
19.52	2	00000000	0D	E7	D4	20	14	EB	00	00	729E	ACK	1
19.52	3	00000000	2D	E2	87	FA	BD	09	BD	03	2963	ACK	2
19.53	4	106FF00C	FF	E2	87	FA	BD	09	BD	03	5A0A	ACK	3
79.73	5	00000000	0D	E7	D4	20	14	EB	00	00	729E	ACK	1
79.74	6	00000000	2D	E2	87	FA	D8	F4	BD	03	5969	ACK	2
79.74	7	106FF00C	FF	E2	87	FA	D8	F4	BD	03	2A00	ACK	3
139.94	8	00000000	0D	E7	D4	20	14	EB	00	00	729E	ACK	1
139.95	9	00000000	2D	E2	87	FA	F2	DF	BE	03	56E9	ACK	2
139.95	10	106FF00C	FF	E2	87	FA	F2	DF	BE	03	2580	ACK	3
199.13	11	00000000	0D	E7	D4	20	18	E7	00	00	2318	ACK	1
199.14	12	00000000	2D	E2	87	FA	0F	C7	BF	03	37B1	ACK	2
199.14	13	106FF00C	FF	E2	87	FA	0F	C7	BF	03	44D8	ACK	3