Rust Implementation of the ANSI E1.31-2018 sACN Protocol

Paul Lancaster

University of St Andrews

0.1 Abstract

The project expands on an existing implementation [2] of the streaming architecture for control networks (sACN) protocol [3] in rust with the aim to make a library that is avaliable in rust that allows ANSI E1.31-2018 sACN data transfer, universe synchronisation, universe discovery and that supports Ipv4, Ipv6, Unix and Windows.

0.2 Declaration

Contents

0.3 Introduction

Introduction Describe the problem you set out to solve and the extent of your success in solving it. You should include the aims and objectives of the project in order of importance and try to outline key aspects of your project for the reader to look for in the rest of your report. At the time of starting this project there did not exist a native, open-source ANSI E1.31-2018 sACN library for rust which allowed data sending, receiving, universe synchronisation and universe discovery. The library that did exist was incomplete and only supported sending data and parsing packets.

This project therefore aims to create a library for rust users which supports ANSI E1.31-2018 sACN sending, receiving, universe synchronisation and discovery. This project is based on the existing library to give a starting point. As sACN is commonly used in heterogeneous device environments with a mix of embedded systems and operating systems such as Windows and Unix to provide support for as many devices as possible a few additional non-functional requirements were made; The library must have support for both Ipv4 and Ipv6 as well as unicast, multicast and broadcast in both windows and unix environments.

The project was split into the following list of primary and secondary objectives.
Primary

Allow sending and receiving DMX data over sACN.

Support the sending and receiving of cross universe DMX data through the universe synchronisation feature.

Support universe discovery with adverts for sources and discovery for receivers.

Secondary

Demonstrate a deployment of the library into a real-world system to show its compliance with the protocol by showing interoperability with other compliant devices.

Provide support for Windows 10 and Fedora Linux systems.

Support multiple IP transmission modes - Unicast, Multicast and Broadcast.

Support multiple IP versions - Ipv4 and Ipv6.

0.4 Context Survey

Context survey Surveying the context, the background literature and any recent work with similar aims. The context survey describes the work already done in this area, either as described in textbooks, research papers, or in publicly available software. You may also describe potentially useful tools and technologies here but do not go into project-specific decisions.

0.4.1 DMX, SACN and ACN

DMX512

DMX512 is an protocol used in the entertainment industry for the control of lighting, effects and other devices. It works by daisy chaining devices together into distinct physical chains (called universes) and is a one way protocol. This means that the devices in the line cannot communicate their presence back to the controller so the controller must know about the devices ahead of time and their addresses so it can broadcast packets down the line which the devices then receive and use. The DMX packets are a fixed size and contain five hundred and twelve 8-byte channel (+ a start code) which allows them to control up to 512 different devices on a singular line. A device may support the use of multiple channels to control different functionalities so for example a light with RGB colour mixing may use 3 channels to allow control of the Red, Green and Blue individually. Since there are only 512 channels available on a single universe this quickly imposes a limitation to the number of devices that can be connected together, especially as modern lighting fixtures commonly use upwards of 30 channels each for a moving light with usage of many more not uncommon. The solution to this was previously to simply have more physical lines (universes) and in this way allow more devices to be controlled simultaneously. This comes with a number of problems however as each new physical line means a new cable coming directly from the control desk.

DMX512 Problems

As the control desk is often far from the devices themselves (at the back of the venue whereas the lights/devices are above the stage) it means that many cables need to be run which can be expensive and time consuming.

The length of the cable runs can cause signal interference / degradation and DMX as a 1 way protocol does not have any error correction (bad frames if detected are thrown out).

The protocol only allowing 512 channels per physical line means that a device cannot have more channels than this. This is particularly a problem recently with the advent of complex fixtures which may have many LED's with individual colour control.

sACN

One solution to solve some of the problems with DMX is to send it using UDP over a standard IP based network and one of the protocols created to do this is sACN. This allows many DMX packets (and so many universes) to be simultaneously sent using a single network cable from the console and then to be received by the devices. Often for backwards compatibility reasons the sACN is converted back into DMX packets before being sent to the device as most devices older than a few years do not support direct sACN communication but this is rapidly increasing - particularly with higher end professional fixtures.

Rust

Rust [6] is a compiled memory safe language with no garbage collector. It is extremely fast with near C/C++ like performance [17] but with a much stricter compiler that guarantees memory safety. As Rust has no runtime due to no garbage collector it is applicable to embedded devices or high performance ..

https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html https://www.techrepublic.com/article/rust-programming-language-seven-reasons-why-you-should-learn-it-in-2019/

Rust [6] is a language designed to be used for writing fast, memory efficient systems code. It has relatively low overhead and because it doesn't require the additional overhead of a garbage collector it is perfect for usage in embedded or high performance systems.

This makes it an ideal language for usage with sACN as responding quickly to control events is vital to a lighting system (imagine a sound effect goes off but the lighting doesn't until a second later).

0.4.2 Related Work

The ANSI E1.31 sACN protocol was originally specified in the document ANSI E1.31-2009 [10]. This represented the base version of the protocol without any universe synchronisation, universe discovery or discussion of operation with Ipv6. Since then it has been revised in 2016 (universe sync and discovery) [11] and again to its current latest version in 2018 (Ipv6). The future of ANSI E1.31 is still being actively developed and discussed [12] with the direction of the ACN eco-system of lighting control data over IP being focused on supporting communication from receivers back to sources.

Within traditional DMX systems this is supported using the remote device management protocol (RDM) as described in ANSI E1.20-2010 [15]. An IP version of the RDM protocol was then created (RDMnet) [13] which is ACN based and allows discovery and control of receivers over a network. RDMnet as a fairly new protocol is still in the process of being taken up by vendors but has strong support from ETC (a large lighting company [16]) in the form of a maintained open source implementation of RDMnet in C++ [14].

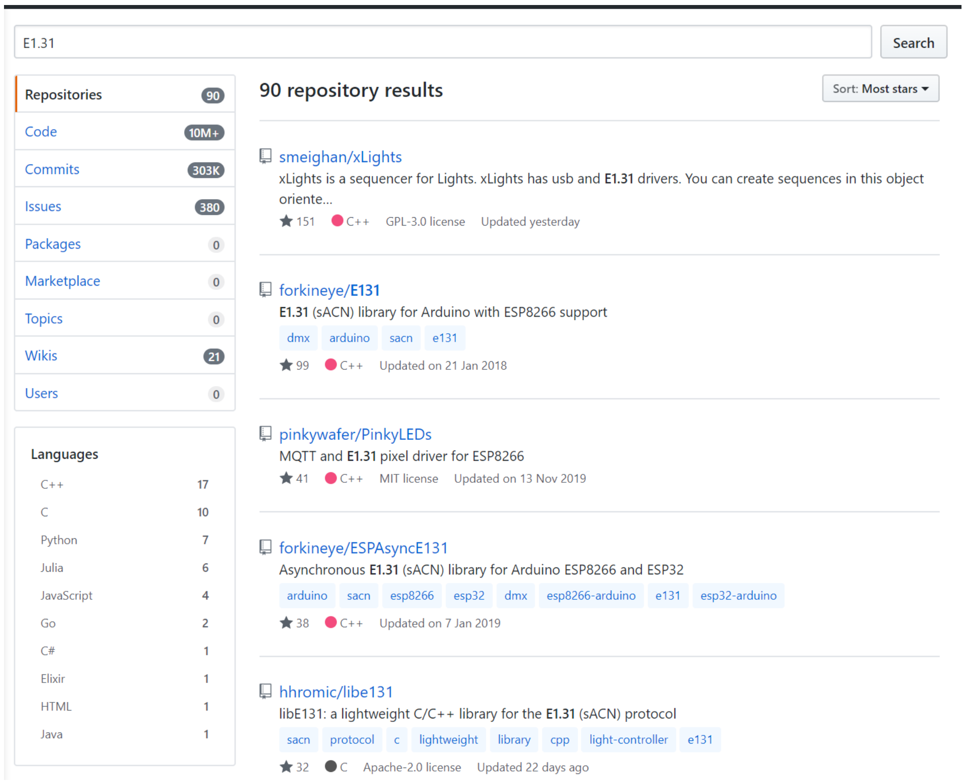
The ACN based family of lighting control protocols aren't the only protocols that allow sending DMX data over an IP network. Another widely adopted protocol is ArtNet which at time of writing is in its 4th version. Unlike sACN on its own ArtNet allows discovery of receivers, remote configuration and transporting RDM data [**?**] in addition to sending data. ArtNet therefore has taken the strategy of being a larger protocol which covers many use-cases as opposed to the ACN strategy of many protocols each doing a specific area that inter-operate. While they are developed independently the ArtNet v4 standard does allow managing sACN devices which means that it can be used to configure/control sACN devices but with the data still sent over sACN [7, Pg. 3].

There are a number of existing implementations of sACN in rust however none are fully compliant with the protocol as specified in ANSI E1.31-2018. One of the most complete is [2] which was used as the base for this project. As this is hosted on github it can be seen that while there are a number of forks (6 at time of writing) no public fork has any further progress which leads to the conclusion that this is the most complete open source rust implementation available. Note that this implementation appears in a number of places such as [8] but this is still the same implementation.

Implementations of sACN exist in multiple languages, at the time of writing (Jan 2020) a cursory search for E1.31 repositories on github reveals repositories in multiple languages with the most prevalent being C++ and C as shown by Figure: 0.4.2. An example of one of these projects is **??** which allows both sending and receiving of sACN packets.

0.5 Requirement specification

Requirements specification Capturing the properties the software solution must have in the form of requirements specification. You may wish to specify different types of requirements and given them priorities if applicable.



Repositories	90
Code	10M+
Commits	303K
Issues	380
Packages	0
Marketplace	0
Topics	0
Wikis	21
Users	0

Languages

C++	17
C	10
Python	7
Julia	6
JavaScript	4
Go	2
C#	1
Elixir	1
HTML	1
Java	1

90 repository results

smeighan/xLights
xLights is a sequencer for Lights. xLights has usb and **E1.31** drivers. You can create sequences in this object oriente...
★ 151 ● C++ GPL-3.0 license Updated yesterday

forkineye/**E131**
E1.31 (sACN) library for Arduino with ESP8266 support
dmx arduino sacn e131
★ 99 ● C++ Updated on 21 Jan 2018

pinkywafer/PinkyLEDs
MQTT and **E1.31** pixel driver for ESP8266
★ 41 ● C++ MIT license Updated on 13 Nov 2019

forkineye/ESPAsyncE131
Asynchronous **E1.31** (sACN) library for Arduino ESP8266 and ESP32
arduino sacn esp8266 esp32 dmx esp8266-arduino e131 esp32-arduino
★ 38 ● C++ Updated on 7 Jan 2019

hhromic/libe131
libE131: a lightweight C/C++ library for the **E1.31** (sACN) protocol
sacn protocol c lightweight library cpp light-controller e131
★ 32 ● C Apache-2.0 license Updated 22 days ago

Figure 1: A search of repositories on github with the search term ”E1.31” as of Jan 2020

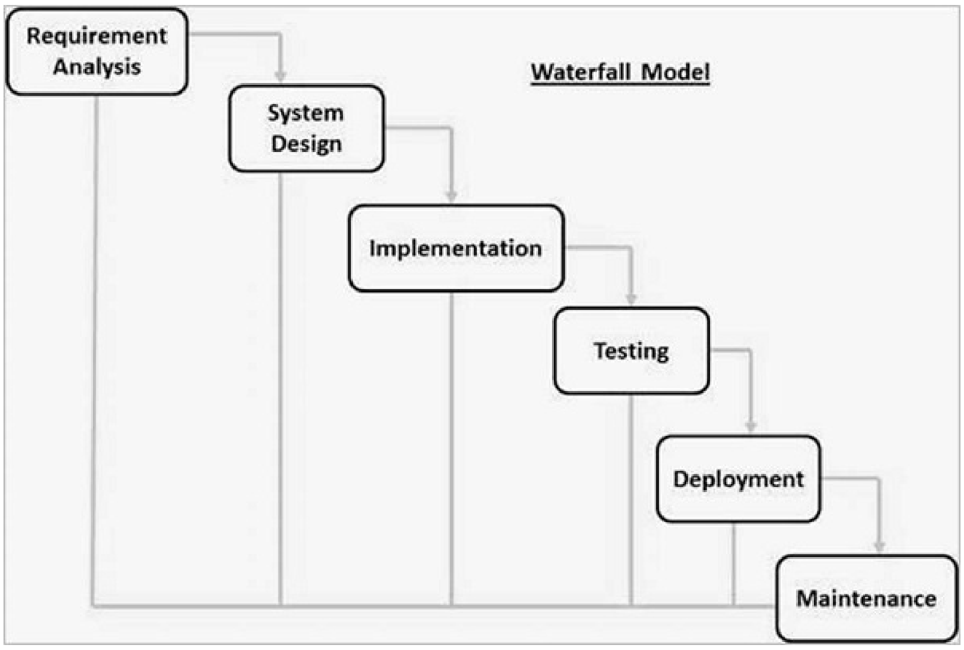


Figure 2: A diagram showing the waterfall development process, [[?]]

0.6 Software Engineering Process

Software engineering process The development approach taken and justification for its adoption.

A waterfall based process model was used for the development of the program. In the waterfall method there are several distinct phases of the project as shown in figure: 0.6. This development approach was chosen as it has a very clear structure which allows easy to manage distinct milestones so progress through the project can be more easily tracked. This process method has a number of disadvantages aswell with the main one being the inflexibility - if something major needed to change it would be difficult to adapt the project. As this project is based on a clearly defined specification provided by the protocal specification and the domains were clearly defined at the start it means that this inflexibility isn't a major issue and so therefore choosing the waterfall method for its advantages makes sense.

The waterfall model can be clearly seen throughout the development of the program. The first phase of 'requirement analysis' is the protocol specification itself as it clearly lays out the goals of the protocol and what it is required to do. On top of this there is the project goals which were defined around the protocol specifically for how much of the protocol this specification should implement for example universe-syncronisation, IPv4/IPv6 support, Unix/Windows support etc. When take together this gives a clear list of requirements as so allows moving onto the 'system design' phase.

The system design phase is where the requirements are turned into a technical plan for how they will be implemented. Alot of this also comes from the protocol specification itself as it describes how each bit of a compliant implementation should behave and so therefore the design can be based of this.

The 'implementation' phase was done
One of the requirements of the project that was defined was that the implementation should be in rust. This combined with the existing base incomplete implementation that was used meant that the general system design was built around this. In general the system was designed around there being distinct receivers and senders with communication being mostly one-way. This meant that the two different sides could be developed in relative isolation as all their communication must be done in a way that is compliant with the protocol which provides the interface between them.

The 'testing' phase is the combining of the various bits of the implementation. This is marked by the passing of the various intergration tests where the sender and the receiver were passing information back and forth. Also part of this phase is the overall compliance tests which are detailed later which show that the implementation of the protocol as a whole conforms to the design and requirements.

Once the testing phase has finished the implementation can move onto the 'deployment' phase. In an industry project this would mean distributing the implementation to users and then later moving to the maintenance stage to fix issues or improve various parts of the project as opportunities or problems are reported. Within this project the deployment took the form of the development of 2 small programs, one which transmits data specified by the user in the form of various dynamic patterns and a receiver which logs all received input. Theses programs followed a waterfall development methodolgy as described later. These programs were then used with various other sACN devices in a real-world environment to show real usage/deployment of the protocol. Issues that are discovered at this point can then be fixed through patches which represent the 'maintenance' phase of the program.

Reflection on the methonolody used In general the approach used worked well for the project as it fit the natural development stratergy meaning there weren't any points where the development felt like it was 'fighting' the approach chosen. There were a few potential problems that were identified however. One of these was that the model forced rigid time constraints.

This is because if too long is spent on any one stage all the subsequent stages would suffer. This was taken as a fairly minor issue for this project because the constraints of fixed submissions deadlines already meant that there some rigid time constraints in place.

?? https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm (01/01/2020)

Ethics

This project has no ethical considerations that require notification in this section.

0.7 Design

Design Indicating the structure of the system, with particular focus on main ideas of the design, unusual design features, etc.

0.7.1 ANSI E1.31-2018

0.7.2 Critique of the protocol

0.8 Implementation

Implementation How the implementation was done and tested, with particular focus on important / novel algorithms and/or data structures, unusual implementation decisions, novel user interface features, etc.

0.8.1 Implementation dependent specifics

0.9 Evaluation and Critical Appraisal

Evaluation and critical appraisal You should evaluate your own work with respect to your original objectives. You should also critically evaluate your work with respect to related work done by others. You should compare and contrast the project to similar work in the public domain, for example as written about in published papers, or as distributed in software available to you.

0.10 Conclusions

Conclusions You should summarise your project, emphasising your key achievements and significant drawbacks to your work, and discuss future directions your work could be taken in.

0.11 Appendices

The appendices to your report will normally be as follows. Testing summary This should describe the steps taken to debug, test, verify or otherwise confirm the correctness of the various modules and their combination.

0.12 Testing

0.12.1 Automated Testing

0.12.2 Real-world Testing

0.13 User Manual

User manual Instructions on installing, executing and using the system where appropriate.

0.14 Other Appendices

Other appendices If appropriate, you may include other material in appendices which are not suitable for inclusion in the main body of your report, such as the ethical approval document. You should not include software listings in your project report, unless it is appropriate to discuss small sections in the main body of your report. Instead, you will submit via MMS your code and associated material such as JavaDoc documentation and detailed UML diagrams

Bibliography

[1] ANSI E1.17 - 2015 Entertainment Technology?Architecture for Control Networks

[2] https://github.com/lschmierer/sacn (September 2019)

[3] ANSI E1.31 ? 2018 Entertainment Technology Lightweight streaming protocol for transport of DMX512 using ACN

[4] https://www.element14.com/community/groups/open-source-hardware/blog/2017/08/24/dmx-explained-dmx512-and-rs-485-protocol-detail-for-lighting-applications (17/09/2019)

[5] https://github.com/hhromic/libe131 (17/09/2019)

[6] https://www.rust-lang.org/ (17/09/2019)

[7] http://artisticlicence.com/WebSiteMaster/User%20Guides/art-net.pdf (17/09/2019)

[8] https://docs.rs/sacn/0.4.4/sacn/index.html (26/01/2020)

[9] https://github.com/hhromic/libe131 (26/01/2020)

[10] https://tsp.esta.org/tsp/documents/docs/E1-31_2009.pdf (26/01/2020)

[11] https://tsp.esta.org/tsp/documents/docs/E1-31-2016.pdf (26/01/2020)

[12] http://www.rdmprotocol.org/files/What_Comes_After_Streaming_DMX_over_ACN_%20%284%2 (26/01/2020)

[13] RDM-NET

[14] https://github.com/ETCLabs/RDMnet (26/01/2020)

[15] RDM

[16] https://www.etcconnect.com/About/ (26/01/2020)

[17] https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust.html (28/01/2020)