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

160007345

January 26, 2020

Rough notes

Title page Containing the title of the project, the names of the student(s), "University of St Andrews" and the date of submission. You may add the name of your supervisor if you wish.

Abstract

Abstract Outline of the project using at most 250 words.

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 fully compliant library that allows sending and receiving DMX payload data without having to handle E1.31 specifics directly. This library is made to support Ipv4, Ipv6, unix and windows and is tested to show compliance with the protocol.

Declaration

If there is a strong case for the protection of confidential data, the parts of the declaration giving permission for its use and publication may be omitted by prior permission of the Honours Coordinator.

Contents Page

Contents

Rough notes 1
Abstract 1
Declaration 1
Contents Page 2
Introduction 3
Context Survey 5
    DMX, SACN and ACN 5
    Other protocols 6
Requirement specification 6
Software Engineering Process 6
Ethics 8
Design 9
    ANSI E1.31-2018 9
    Critique of the protocol 9
Implementation 9
    Implementation dependent specifics 9
Evaluation and Critical Appraisal 9
Conclusions 9
Appendices 9
Testing 10
    Automated Testing 10
    Real-world Testing 10
User Manual 10
Other Appendices 10

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.

The goal of this project was to create a rust library for sending and receiving sACN data that is fully compliant with the protocol specification as defined in ANSI E1.31-2018 [3]. This project is based on an existing but incomplete implementation of the protocol [2] which was then expanded upon.

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

Learn rust - This is an inherent part of the project as rust is a new language.

Establish the extends of the existing implementation - Only some parts of the protocol had been implemented so need to establish which bits and to what extent.

Introduce more through testing of existing - There is only a small amount of testing for the existing implementation which needs to be expanded on to verify it works

Allow receiving DMX through sACN

Allow sending DMX using universe synchronisation - This involves both data packets and sync packets.

Allow universe discovery - Sources of data can be discovered using adverts.

Add support for the options field - This includes things such as termination of streams and forced synchronisation.

Secondary

Analyse the performance of the protocol implementation verses other implementations in other languages - Using metrics such as data throughput, performance as number of universes increases.

Analyse the performance of the protocol versus other similar protocols such as ArtNet [7]

Test the implementation in a real-world environment with real devices - Involves creating a small demo program.

Windows and Unix support - Windows 10 and the lab machine Fedora version.

Unicast, Multicast and Broadcast Support - The protocol works over all these with multicast being the preferred.

Ipv4 and Ipv6 support

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.

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 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).

Other protocols

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.

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: . 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

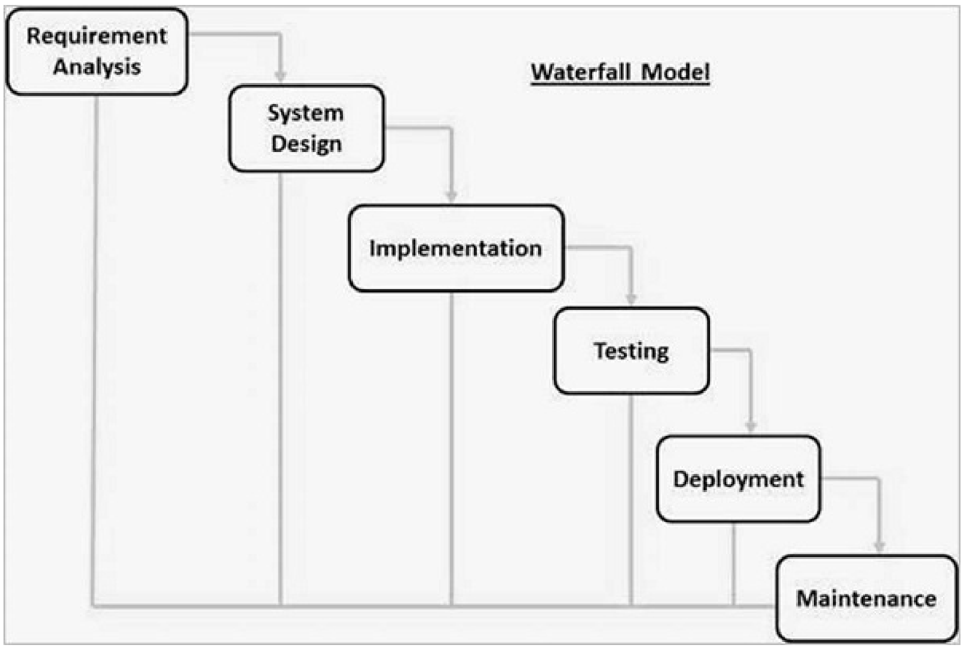

Figure 1: A diagram showing the waterfall development process, [[?]]

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

Ethics Any ethical considerations for the project. You should scan the signed ethical approval document, and include it as an appendix.

Design

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

ANSI E1.31-2018

Critique of the protocol

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.

Implementation dependent specifics

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.

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.

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.

Testing

Automated Testing

Real-world Testing

User Manual

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

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)