CS4099 DOER: Rust implementation of the sACN protocol

Paul Lancaster

September 23, 2019

Description

Aim

The project aims to expand an existing implementation [2] of the streaming architecture for control networks (sACN) protocol [1] in rust so that it fulfils the scope of the protocol as defined in [3]. While open source implementations of the sACN protocl exist in other languages [5] there are no fully implemented versions in rust (as of starting this project).

Background

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).

Objectives

Primary Objectives

1. Establish the extends of the existing implementation and introduce more thorough testing to verify its correctness (fixing issues found).

2. Expand the existing implementation to allow receiving sACN.

3. Expand the existing implementation to allow sending DMX with universe synchronisation.

4. Expand the existing implementation to allow universe discovery.

Secondary Objectives

1. Analyse the performance of this protocol implementation verses other implementations in different languages using a number of different metrics.

2. Analyse the performance of this protocol in various implementations verses another protocol such as ArtNet [7].

3. Make an alternative implementation of the sACN reciever to use non-blocking or blocking IO (depending on which wasn't used previously).

4. Test the completed implementation in a real-world environment by creating a demo program which utilises the library.

5. Window and Unix support.

6. Introduce unicast and broadcast support.

Tertiary Objectives

1. Ipv4 and Ipv6 support

Ethics

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

Resources

For testing the protocol and verifying it works with real world devices this project will require access to a control device capable of generating sACN packets and a device able to receive sACN packets and use them. I have access to these devices already as well as all equipment needed to connect them and so I do not require any equipment from the CS department.

Bibliography

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

[2] https://github.com/lschmierer/sacn

[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)