# VIDEO OVER IP FOR THE BROADCASTIGN INDUSTRY

Computer Science Co-op Student – Work Term 1

Software Engineer with Matrox Electronic Systems Ltd.

Submitted to Dr. Rajakopalan Jayakumar

By Christopher McArthur (40004257)

Concordia University December 6,2016

Table of Contents

**Submittal Form**

| | |
|---|---|
| **Student ID** | 40004257 |
| **Student's Name** | Christopher McArthur |
| **Co-op Program** | Computer Science |
| **Work Term Level (1,2,3,4)** | Level 1 |
| **Work Term / Year (Fall, Winter or Summer)** | Winter 2016 |
| **Organization/Dept.** | Matrox Video |
| **Address** | 1050 St Regis Boulevard Dorval |
| **Supervisor's Name** | Sylvain Mailloux |
| **Supervisor's Email** | smaillou@matrox.com |

| | |
|---|---|
| **Title of Report** | VIDEO OVER IP FOR THE BROADCASTIGN INDUSTRY |
| **Date Submitted** | December 9th 2016 |

**Work Term History**

| Work Term Level | Employer | Position held | Semester and Year | Remarks: *International work term, change of major, back to back internship etc.* |
|---|---|---|---|---|
| 1 | Matrox Electronic Systems | Software Engineer | Winter 2016 | First work term |
| 2 | | | | |
| 3 | | | | |
| 4 | | | | |

**Acknowledgement of assistance in the preparation of this report (if applicable):**

**Student's Signature:**

## 1. Abstract

My internship at Matrox consisted of work on an inter-process communication API which would work between Matrox's main software solution, Topology Builder, and a daemonized service which would be a software IP stack running on top of the X.mio3 IP card. Working from start to finish on this project, I commenced with learning the socket layer, followed by wrapping it within a process channel layer than implementing that to the functionality of four services needed to preform video over IP. I developed my skills as a programmer, learning how to apply advanced coding techniques and produce code for a professional level. I also how the luxury to gain insight into software design and architecture. There's was minimal usage of school material as what I works on was far more advanced thus I relied heavily on my knowledge which I gained form personal projects. I will continue my learning by further developing skills regarding memory sharing and networking with the intention of developing my own server and client periscope.

## 2. Overview

The result of my experience at Matrox was more than I expected out of a first internship; above working at the job I dreamt of for the last four years, the team which I was placed with was a perfect fit. It has been a humbling experience being with two individuals with a combined experience exceeding 35 years in software engineering; ting my skill set to shame and rightfully so. I wrote simple classes recreating an existing design patterns while my coworker created dozens of classes, wrapped serval existing technologies and implemented them in an agnostic and cohesive manner to be implemented on top of hardware. Being a self-taught programmer has its pros and cons, for example I know everything, another example is I know absolutely nothing  when it comes to doing it the right way. This internship has allowed me to grow and develop my skills on a professional and technical level. All my success and work was related to the great support from my team. My internship started with investigating a bug which was left unresolved. I worked in iterations and developed an inter-process communication API that was the foundation to 4 services. Towards the end of my stage I work on a bug where a progress bar did not progress. There were concepts taught in school which applied but I mostly relied on my personal experiences and knowledge to drive my work. There was also the discovery that I am a very solid employee which an employer could appreciate.

### 2.1 The Video Process Team

I was a part of the video departments Process Team, whose main function is to maintain all the tools and services which support the development on software, this is namely a build system consisting of a server (build server), agents (application running on everyone in the department's computer that would build projects in the background and upload them to central database), and build machines (both Windows and Linux, top tier machines to handle complex

large projects). The secondary functionality, which had recently been added in the last year, is the development of software for the Matrox SDK which is sold to customers who use Matrox's hardware to make their own software modules. The main task assigned to the team was the development and production of a software which enabled video over IP though a new card, the X.mio3 IP, produced by Matrox. The team consisted of two others 'my supervisor' and 'my coworker'. Being a small team we worked closely with one another, helping each other navigate difficulties, bugs, linker issues and design challenges. The team was structured off the SCRUM datagram and used the agile method to outline our workflow. We worked in three week sprints, with daily meetings and would sit at the midpoint to discuss our sprint and make sure we were on track.

My integration into the team was seamless, my supervisor and coworker have been working together for several years now developing the build system and developed a good working relationship, but they have very different personalities and I was the middle piece which balanced and improved the work environment. This is one of the main factors my internship was so pleasant, having a great relationship within the team made working together extremely profitable for us and the company.

### 2.2 Video over IP

The main project the team was assigned during my internship was the development of software that utilizes the new X.mio3 IP card and allowed for TCP and UDP packets to be managed in an IP stack. These would be running as a daemonized services which would be accessed Through the main software Topology Builder and the SDX core offered by Matrox. This software is meant to be implemented within

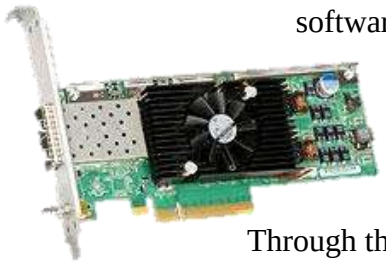

*Figure 1 - The X.mio3 IP card*

the broadcasting industry as a part of the shift in technology from the existing limited SDI cables to 10 gigabyte IP network cables allowing for remote production of material.[1] (EBU technology & innovation - live & IP production, 2016) The standard SDI cables can only carry a signal for up to 100m for HD whereas with IP the signal can be sent 10s of km within an instant. (Laabs, 2012)

### 2.3 Inter Process API

My role in this project was the development of the tool which would allow for communication from the Topology Builder application to the daemonized service which will interact with the card; an inter process communication API was need to allow for this to happen. The project required that it be cross platform for Windows and Linux, which imposed limitations and extra challenges onto the nature of the project. Limitations which applied to my personal involvement was that I never needed to program in C++ pre 2011 when a huge set of tools were released within the standard library.[2]

### 2.4 Bugs

As all software engineers I was also tasked with fixing bugs, a Linux clock notification error and a Progress bar which showed no progress. The Linux clock issue was a huge challenge as it was my first task, though I found the cause of the problem, my discovery of more bugs led to the clock notification remaining unresolved. My second bug was a progress bar on driver

[1] There was a guest speaker who is the Senior project manager for this experiment, he was going to New York's broadcasting industry's annual meeting to present the innovative work he was doing, some of the equipment that is used is in part made by Matrox. It was nice to know what I was working on was the future of how the TV we watch is produced.

[2] It's actually shocking how hard it is to multi-thread (mutexes, semaphores, condition variables) using VS2010 … it was actually easier to do thinks on Linux than it was on Windows at first; there was a hard learning curve to understand Matrox's internal software. It was on my teams mandate at the begging of my stage to upgrade to VS2015 but sadly that was put on hold for the X.mio3 video over IP project.

install which did not progress, this ended well after finding a commented out line which was necessary to allow it functioning correctly.

### 3. Universal Clock Notification Not Displaying in Linux

My first bug and assignment in my internship was a handful. What exactly is a universal clock? I had three hours of experience using Linux. I have not coded in C++ in serval months. In short, this bug was pushed back down the backlog after I discovered new bugs, sadly I was unable to complete this task. I began by reproducing the bug on the Linux workbench, which was very simple and was a great refresher to command line which I only ever used with Git. I took me about a week to understand Matrox's very outdated coding convention and learn how this program worked. Once I got my feet wet and was comfortable with what I was investigating, I moved on to using a software called Wireshark to trace the packets. Initially the clock sent a burst of NTP time packets. Than the clock updated its calculation at a certain frequency by sending a new burst NTP time packet. Now when the investigation returned to testing on the Linux, I uncovered that the system was sending packets in insufficient quantities and the clock was not able to perform its calculation due to missing packets, thus not displaying the expected notification. The next step is to debug to find the problem, so my supervisor started me off by installing a virtual machine so that I can test. He partially installed the VM in the root, instead of a subdirectory, which corrupted the OS. Now I was tasked with installing and configure Linux Mint 17.3 for a professional use with things like mounts, remote access, virtual machines and packages. It took me three days, but I am very capable at Linux and the command line. Now returning to the problem with debugging the app on Linux, there was an internal 'Debug Session' available with the build system. Now this works and has been used plenty on LinuxAgents (build machines). Yet when I tried to start one on the workbench, nothing happened. For whatever

reason the ‘Debug Session’ terminal was never launched. I had the luxury of entering a story into our SCRUM backlog. The bug was put aside and was never resumed during my stage, remaining uncompleted.

## 4. Development of the Process Channel Layer

My stage was primarily focused on the development of an inter-process API using socket based technology and a library that was open source. The CSimpleSocket library[3] was developed to provide a cross platform, Windows and Linux, access to sockets and each OS has a different diver implementation within their respective kernels. This library was partially used previously in Matrox before and had begun to be standardized to the SDK coding convention. Doing so was a great stepping stone to learn to read and write code as there is a huge change in mindset to implement code for others to use not just yourself. The initial design was devised by my supervisor before I began working on the project. Following several iterations, it went from A simple inheritance tree to sophisticated design using objects to their fullest. The first step began with a test app to modify the CSimpleSocket library to reflect Matrox's workflow of “event check result did succeed than continue” a very intuitive way because it prevents running anything further than the failed function. What this meant for me was updating all the function

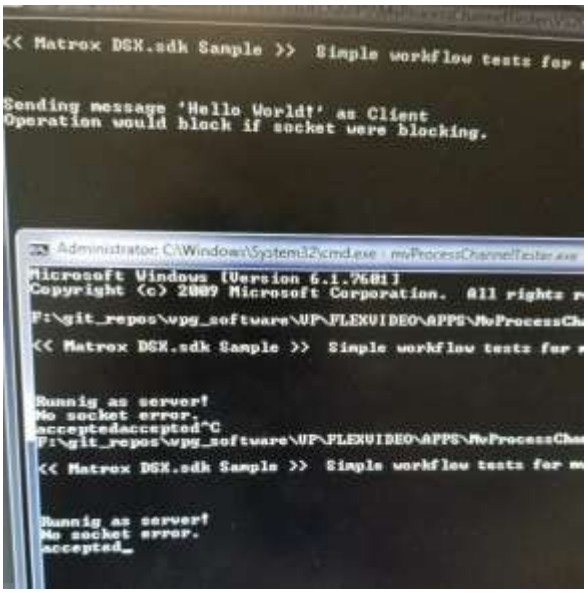


*Figure 2 – Active socket being accepted by a Passive socket*

[3] This is a public open source library which can be found on github (https://github.com/DFHack/clsocket). It’s a cross platform wrapper which handles all the #ifdef LINUX #else fr Windows.

calls I was using to return an HRESULT. The CSimpleSocket library had three main objects the simple, passive, and active sockets. A passive socket represents a server while the active represents a client. The server binds his address to a port and listens for new connections. A client attempts to connect to the server port and address, passing its information through the socket once a connection is established between them. The server calls accept creating a pointer to the client it just accepted. Now both sides are free to send and receive as the wish. In typical programmer style my first message to be echoed by my server was hello world.

With the completion of learning the primary library, my work began to a beginner level, I now needed to create tree classes. A base ProcessChannel class which would lay the roundwork for the common functionality like initializing, sending, receiving, and disconnecting. In addition to this, there would be ProcessChannelClient and ProcessChannelServer classes to handle each side's individual functionality and derive from the base class.

### 4.1 The ProcessChannel Class

This base class has a singular goal which is to define everything that is common to both the server and client partitions. This includes an additional set of methods for reading and writing the registry which was need for the shared knowledge of the port to use and for the clients need for the address of the server. Never having used the Windows registry my supervisor sent me a .htm page he wrote on Windows 95 about how the registry worked, and it hasn't changed one bit since then! The downside of this design is that Linux doesn't have a registry, lucky for me Matrox had an internal class which handled both the Windows registry and Linux keyring so much work was just learning how to use this encapsulate class. For communication at the socket level three main functions were needed, SendMessage, ReadMessage, and PostMessage. SendMessage writes an 8-byte buffer to the socket, ReadMessage will extract the 8-byte content

form the socket but will block if there is nothing to read. Post message is send and read message in one for a synchronous call. Initialize, Disconnect, and WaitForStatusChange are all socket maintenance functions for more stable infrastructure.

### 4.2 The ProcessChannelClient Class

This class introduce the connect function which allows an active socket to open a communication channel with a passive socket by providing a certain address and specific port. Its first prototype sent the famous "Hello World!" And listened for a response.

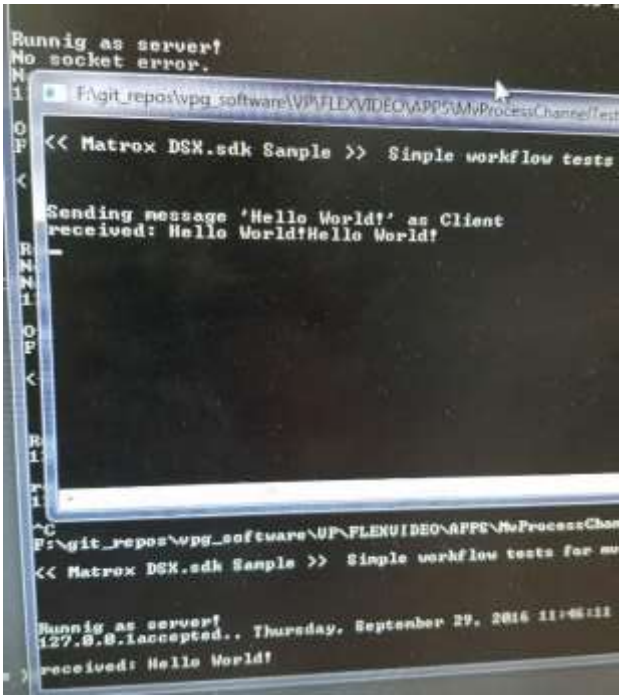


*Figure 3 - First message, "Hello World!" successfully being transmitted*

### 4.3 The ProcessChannelServer Class

This class was prototyped as an echo server, whatever message it received will be resent to its connection. It was blocking and could only interact with a single client before connecting with any other. It held the Bind and Listen functions, responsible for write its information to the socket than opening it for connections to be established respectively. Following theses was the accept function which returned a pointer to the new connection represented by an active socket.

## 5. Introduction of the Four Servers

The motivation behind the entire inter-process communication API which I was tasked with was for the management of four servers all which were to take place through the networking card. For video over IP, the broadcasting industry was using the SMPTE

specifications which define the usage of PTP packets for timing and UPD or TCP video packets. (Society of Motion Television Engineers, 2016) In addition to this, Matrox's client(s) had addition requests with required handling TCP connections. All of the internet protocols and packets would be passed from the buffer of the card to a software IP stack. It was my inter-process API which would take the clients request from the software GUI Topology Builder to the IP stack running as a separate process. The children class of the ProcessChannelClients would be under the hood of the Topology Builder application[4] while the ProcessChannelServer would be daemonized along with the IP stack to interface with the card. Video over IP within consists of four main sections, joining a multicast group to receive video feed(s), the SMPTE specification for timing, managing network settings, and controlling who can access your database. None of these were ready for implementation so my worked consisted of skeletons following an interface with the exception of IGMP used for joining and leaving multicast groups. (He, Sandick, and Haberman, 2006) Each came with a client and server side with a middle class where everything that was common to its functionality would go.

### 5.1 The SocketMessage Object

There needed to be a standardized way to format all message so that processing would be efficient and simple to implement for no matter what service may need to be implemented or added later. My supervisor showed me the message object which was used in the build system between the server and agents. Unfortunately, it wasn't made to run on Linux so it took some work changing all the types to C++ or STD types and ensuring everting was properly format.

[4] This is the main software which Matrox sells which allows one to test different configurations of video processing schemas with the SDX Core on top of their cards.
(http://www.matrox.com/video/en/products/developer/software/topology_builder/)

There as a very complex reading the socket buffer which took some navigating but I utilized it in the ComposeMessage function which I added. The message took an ANSI encode start/stop and parameter separators. It took a service code, operation code followed by a variable number of string parameters; it took quite a while to have all of this working properly as its need to change types that need more than casting allowing if a large of amount of logical operations. Upgrading the Process channel layer was not so bad though, after receive a string from the command line instead of writing it to the socket it would format a service code, operation code and attach the string each word being a parameter. It would than convert the message back to a string and write it to the socket, all of this was done within the SendMessage and ReadMessage so that within the code all that was seen was the Message object.

### 5.2 The Servers definition

This class changed dramatic through the iterations of the project but at first they started off instantiating a ProcessChannelServer which listened to the socket, with a new connection passed to a thread. Than processing would start; reading would extract the service code from the message, translate that with a string compare to a known integer value and switch to perform the proper child server's functionality. This held several major design flaws; a base class should not be performing operations specific to its children classes. This was a flaw which had arisen in the iterative workflow and was not properly designed on my part. A step which was required was to turn the .exe to a .lib so that other projects could depend on it which was not possible. Yet another problem was that only one Server could exist as only one passive socket can bind to a socket. A major redesign was needed in order to fulfill the requirements. Even the message object was flawed it used definitions in three different locations for service types and another two locations held the definitions of operation codes which lead to many hours wasted.

## 6. The Final Product

There was a complete rework of the whole entire Server side, in fact the ProcessChannelServer became ProcessChannelService and has no functionality what so ever in regards to the passive socket. ProcessChannel changed to what was common between a client and a service provider and its functionality for sending and receiving was move to Connection. ProcesssChannelClient improved to be more efficient. The ProcessChannelCenter was the new "server" object, it created ClientConnections which handled the reading and calling the right service to ProcessMessage which was a pure virtual of the ProcessChannelService. Lastly, the SocketMessage was improved to better standardize service codes and operation codes.

### 6.1 Connection Class

The Connection class took over the old ProcessChannel. In Visual Studio it was simply a rename and the ProcessChannel died for two weeks. This new class held all of the definitions of SendingMessage, ReviecingMessages, Writing/ReadingTheIpFromRegistry, Writing/ReadingThePortFromRegistry. In Addition to theses there was also the socket maintenance functions Initialize, WaitForStatusChange, Linger and Disconnect with the set of methods to access the registry.

### 6.2 The ProcessChannel Class

This class was erased from existence for a while through the redesign as it was no longer needed with the creation of the Connection class. Its exuberant return was to the required of a ServiceID which both a client and service would need to identify themselves with. This identifier would be unique between each service and client pair. The ServiceID was used in the ClientConnection (server side perspective) when a client's message needed to be sent for processing, a map was held with all the service providers and their ServiceID. There needed to

be a match in order for processing to be called otherwise the client would be notified that there was no service provider for its ServiceID.

### 6.3 The ProcessChannelClient

This class became an interface with two pure virtual members, one being Initial the other being Disconnect.[5] These calls were implement to establish or close the connection at the application level. For example, calling Initialize would, setup the active socket, connect to the passive socket and subscribe for services. Upon returning from this method the client would be primed for requesting task or information from its.

### 6.4 The ProcessChannelService

This class became a shell of its former self losing all of the passive socket and server functionality. It had a single a pure virtual method ProcessMessage which its children implemented as the main functionality; reading the message it would check the operation code, perform the correct task(s) for that specified operation. There would be a response written so that the client would know what the result of its request is. This child was completed for IGMP which would check the operation code, validate the multicast address given than use the IP stack to send an IGMP join request to the requested multicast. It could than send the IGMP leave, perform the same validation of address and have the IP stack send the IGMP leave. This was tested and worked for a video feed.

### 6.5 The ProcessChannelCenter Class

This is where the server now resides. It holds a map of ServiceIDs to ProcessChannelServices, a double ended queue of ClientConnections for memory management

[5] I just realized in reviewing this that I'm most likely going to change these as they are all implement in the exact same manner for each child class.

and deletes everything in the destructor. There are two public functions which are used in the daemonized service. The RegistereService function takes a ProcessChannelService and extracts its ID and adds both its to the map. The Second call is Run, which preforms all the necessary steps for the socket than launches a thread which will wait for new connections. It loops until the KillEvent is set. The Accept function at the socket level is non-blocking so the while loop has a 20 millisecond delay so not to waste clock cycles, pooling isn't the best design but that's the only way to implement this library. Upon accepting a new connection, the ClientConnection is created and handles the rest of the server side functionality.

### 6.6 The ClientConnection Class

When this class is instantiated, it takes an active socket and a pointer to a map of IDs and Services. In its constructor it launches a thread that will ReadMessage, once it receives a message from its client it will get the service code of the client from the message and check the map for its paired service, if no match is found a message will be returned to the client to notify them of such an event, otherwise the proper service will be used to call ProcessMessage. Once the processing is over the reply message generate by the processing will be sent to the client informing it of the result of its inquiry, the loop will than restart. The loop will terminate if the client disconnects or is delete otherwise it can be shut down if the KillEvent is set by the destructor of ClientConnection.

### 6.7 The SocketMessage Class

This was slimed down and stripped to be a lean and mean message object. The service codes and operation codes were all defined singularly in the ProcessChannel class in two Enumerators; the message was now composing with these as numeric integer values transformed

to strings. Words were no longer stored as individual parameters but the variable number of parameters was kept and implemented properly.

### 6.8 Linux Excitements on the Inter-Process API

I started my stage installing Linux and I worked on it the whole time. At the end of each SCRUM sprint the last thing was always trying to get all the new changes to compile on Linux so that the story could be closed. Using LinuxAgents and the Debug Session it was always the last rush. I can write my own Makefile for a project, I actually have sufficient understanding of Linux and compilation to do that now. It was actually nice because I kind of enjoy working on Linux now. This is because I love a challenge and learning something new; Visual Studio does so much work for you that the command line gcc/g++ compiler does not and gives horrible error messages which are hard to find solutions to when googled.

## 7. Progress Bar Not Updating on Drive Installation

This was the second bug I worked on and it was super easy! By this point I was completely in the swing of looking at Matrox code, hell I was writing it, but I have had personal experience with doing progress bars on windows. My first day on this bug was utilized learning the code, the individual who was responsible to its deployment was in so the second day with his help I was able to trace the code having provided the proper arguments. Driver installing varied by each different card, the particular card my bug was for was installed by .inf files. After tracing the code, I released that the line setting the value to zero was commented out so when the calls were made to increase the bars value it was unable since it was never set to zero. To my success I closed the bug and I have a commit on develop (the master branch).

## 8. Reflection of Theory and Practice

There are plenty of concepts taught in school which I used, C++ is an object oriented language so I did apply inheritance, encapsulation and interfaces into my code, but I cannot say I learned those at school. I started programming in C++ at the end of high school by college I was writing multithreaded windows GUIs selling my programs online. In school we saw LISP and PROLOG, but should those be replaced with languages like Batch, Bash or Shell; we should learn what a shebang is and why it's there. Next semester I have 5 computer classes lined up for myself include Advanced C++ but I'm sure all the memory management, multithreading, mutexes and semaphores I used will be lucky to be placed in the course material. I was very fortunate to have had the opportunity to learn how to implement theses at home in my personal projects otherwise I would have been lost in this internship.

## 9. Reflection as an Employee

I am surprisingly a great employee; usually I have problems with authority assigning me task but really not in the case when its more my supervisor giving me tasks where meant for learning how to code better than what I knew. I held great respect towards my supervisor and coworker, they were always knowledgeable and more than willing to help. I didn't pretend to be someone I am not, I was slightly on the forceful side when giving my opinions and I expressed myself and gave my input on my project; nonetheless I still held my position though, I was the intern there to learn even if I thought changing my project more was not necessary, I still did my best work because at the end of it, no one will be coming to see me questioning why I made certain design choices, it's my supervisor who will be held responsible, he had the final say.

## 10. Conclusion

My internship was a very positive experience due to great chemistry within the team and a challenging work environment. Working through an iterative cycles following the SCRUM layout and agile manifesto, there were several evolutions in the development of the inter-process communication. Beginning at the base socket I learned the fundamentals of the technology. From there I developed base classes which held all the functionality. When the introduction of the four services highlighted the flaws in the design. The final product was a well-defined object based which meet all the criteria of the project. My work on the development of an inter process API solidified my knowledge of programming principals such as multithreading and memory management. Additionally, I gained hands on experience with software architecture, implementation of design patterns as well as how to write code in a professional environment. My personal knowledge of programming was the base of my work and grew in parallel with the complexity of my work. In the end I fulfill my rolled as an intern doing exemplary work and completing the task while learning how to produce code at a higher level.

## 11. Recommendation

As a first year student, I found the material be slow paced in its progression. There are many languages which would provide more relevant knowledge in regards to applicable in the work place. In a course about programming principles, introducing scripting languages like batch, bash or shell would help gear students for working in the real world. Teaching higher level languages which I more relevant than Java, require students to develop more in-depth critical thinking and analyses with pointers, references, memory management. Students applying for this position should be prepared with a working knowledge of C++ and understanding of the Linux kernel. The Co-op internship program is well structured and provides valuable support and

resources. The institute can improve its service by using some of its potential interns to provide web development services to allow for compass to be accessible on mobile phones.

**Table of Figures**

## References

Laabs, M. (2012) ‘SDI over IP - seamless signal switching in SMPTE 2022-6 and a novel multicast routing concept’, (1609-1469).

European Broadcasting Union (2016) *EBU technology & innovation - live & IP production*. Available at: https://tech.ebu.ch/live-IP (Accessed: 3 December 2016).

He, H., Sandick, H. and Haberman, B. (2006) *Internet group management protocol (IGMP) / Multicast listener discovery (MLD)-based Multicast forwarding (‘IGMP/MLD Proxying’)*. Available at: https://tools.ietf.org/html/rfc4605 (Accessed: 3 December 2016).

Society of Motion Television Engineers (2016) *SMPTE® Publishes First Two Parts of Standard Enabling Deployment of PTP-Timed Equipment in Existing SDI Plants*. Available at: https://www.smpte.org/news-events/newsreleases/smpte-publishes-first-two-parts-standard-enabling-deployment-ptp-timed (Accessed: 4 December 2016).