A Proxy Solution for Networked Audio Device Interoperability

Osedum P. Igumbor, Richard Foss
Department of Computer Science, Rhodes University, Grahamstown 6140, South Africa
Tel: +27 46 603 8291, Fax: +27 46 636 1915
Email: g06i3351@campus.ru.ac.za; r.foss@ru.ac.za

***Abstract*—This paper proposes the use of a proxy as the means to achieving network interoperability. This research investigates whether proxies can be used to attain cross network communication. In this paper two protocols are described, namely the Audio Video Control (AV/C) protocol and the contemporary AES-X170 protocol. An AV/C proxy that understands both AV/C and AES-X170 is developed and tested for connection management and device control.**

Presently there exist numerous audio devices that conform to legacy protocols such as AV/C in commercial use. The challenge is to integrate such devices into modern professional audio networking solutions such as AES-X170. The proxy has been developed as a high-level (application level) solution to network interoperability. This ensures that the internal details of a device in terms of its address mappings and low-level functionality do not restrict device interactions.

By implementing a proxy solution for network interoperability, consumers can now take advantage of modern networking solutions by integrating their legacy devices with devices that implement more recent control and connection management protocols.[1]

***Index Terms*—Audio Video Control (AV/C), Ethernet, firewire, Internet Protocol (IP), OSI/ISO, AES-X170**

I. Introduction

THE networking of audio devices requires that a physical connection is made between devices, and communication between the networked devices is established according to a protocol. In a digital audio network, devices can be physically connected using physical network connections such as Ethernet, IEEE 1394 serial bus [12] (hereafter referred to as 'firewire') or universal serial bus (USB). Each of these physical mediums has its merits. Presently, there exist several protocols that enable communication between networked audio devices. These include:

- HiQnet - a proprietary control protocol developed by the Harman Pro Group. It allows for device configuration, connection and control using its application interface called HiQnet System Architect [17].
- CobraNet - a protocol for real-time audio signal distribution over Ethernet at relatively low-latency [8].
- AES24 - a standardized peer-to-peer protocol for controlling and monitoring networked audio devices. The peer-to-peer attribute of AES24 ensures that any device on the network is capable of transmitting and receiving instructions from other devices on the network. AES24 models audio devices as discrete functional objects based on a 4-level hierarchy of objects, devices, subnetworks and internetworks [7].
- AV/C - a standardized command and response protocol for networked audio and video devices. Audio Video Control (AV/C) protocol is a firewire command/transaction set that uses the Function Control Protocol (FCP) defined in IEC 61883-1 (2008) specification [4].
- mLAN - a network solution for audio devices developed by Yamaha Corporation. The music Local Area Network (mLAN) version 2 implements an Enabler/Transporter ('*Plural Node*') architecture where the Enabler resides on a PC and handles connection management. A transporter is implemented within an audio device and is responsible for the transmission and extraction of isochronous packets in accordance with the IEC 61883-6 (2005) specification [16].
- OSC - Open Sound Control is a message-based protocol for multimedia devices. This protocol defines an *OSC client* that sends an *OSC message* to an *OSC server*. The OSC protocol is a (OSI/ISO) transport layer independent communication protocol [19]. It uses a URL/directory style symbolic addressing scheme, and provides *'wild card'* symbols to specify multiple destination addresses [18].
- AES-X170 (trade name XFN) - a peer-to-peer command and control protocol for networked audio and video devices. It uses the Internet Protocol version 4 (IPv4) for the transfer of AES-X170 messages between devices that implement the AES-X170 stack [9].
- ACN - Architecture for Control Networks is an Ethernet-based multipurpose network control protocol suite. This protocol is an application layer protocol being developed by the ESTA (Entertainment Services and Technology Association).

Some of these protocols are proprietary (for example HiQnet) and may require that only devices that conform to certain hardware manufacturer defined specifications can implement them. An even greater challenge with these protocols is that a device will typically implement only one of them. This is due

[1]This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University, with financial support from Telkom SA, Comverse, StorTech, Tellabs, Mars Technologies, Amatole Telecoms, THRIP and the Department of Labour (DST) SA.

to the cost implications of implementing multiple protocols on a single device. As a result devices that conform to any one protocol are unable to communicate with devices that implement a different protocol. The implication of this to the consumer is that they are restricted in the type of devices they purchase, since they have to ensure that any new device complies with the protocol of their already existing network.

This research investigates two protocols (Audio Video Control (AV/C) protocol and AES-X170 protocol) in an attempt to provide a solution that allows devices to communicate across both protocols. The AV/C protocol has a wide range of application in professional audio networks, and AV/C devices are commercially available. The choice of the XFN protocol was based on its accessibility to the researcher who belongs to the Audio Engineering Research Group at Rhodes University, which is involved in the standardization of the AES-X170 protocol.

In the following sections AV/C and AES-X170 protocols are discussed.

A. Audio/Video Control protocol (AV/C)

The AV/C protocol is actually a transaction set that defines various commands and responses that can be used for communication between firewire networked devices. Firewire is a high speed serial bus that allows for asynchronous and isochronous transactions [6]. Firewire is capable of transferring data at speeds of 400Mb/s, 800Mb/s, 1.6Gb/s and 3.2Gb/s [5]. The asynchronous nature of firewire guarantees delivery of transmitted packets on the bus by ensuring that all transmitted packets are acknowledged. The isochronous nature of firewire ensures consistent transmission of packets. On a single firewire bus 63 nodes can be networked, and as many as 1023 buses can be connected on a bridged network. A node represents one firewire card that interfaces with the bus. While firewire provides other advantages such as bus management and plug and play, one of its most appealing attributes is the possibility to transmit asynchronous and isochronous packets on the bus over a 'cycle period'. Typically, asynchronous transactions are used for the transmission of control messages while isochronous transactions are used for the transmission of real-time media such as audio and video.

Firewire allows for direct memory addressing, this means that firewire devices expose internal register space through their firewire interface. Firewire uses the 64-bit Control and Status Registers (CSR) architecture addressing scheme defined in ISO/IEC 13213 (1994)[10] to identify devices on the network [6]. In this addressing scheme, the most significant 10-bits are used to specify the bus. Following this is a 6-bit field used to identify the node, and the least significant 48-bits is used to address the memory offset within the identified node. The layout of this addressing scheme is shown in the Fig. 1.

The IEEE 1394 serial bus provides a networking technology that forms the basis of the IEC 61883-1 (2008) specification [11]. The IEC 61883-1 specification defines a Function Control Protocol (FCP) that depends on direct asynchronous writes to a device's memory address. In FCP a controller node sends

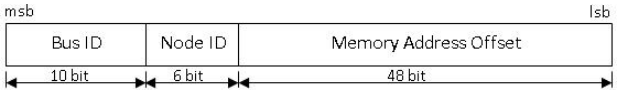

Fig. 1. Layout of the CSR 64-bit addressing scheme

an FCP command to the FCP_COMMAND register within a target node. This triggers an automatic FCP response from the target node to the controller node's FCP_RESPONSE register. Every device that conforms to FCP is required to implement these two registers, namely; FCP_COMMAND and FCP_RESPONSE registers.

There exist various AV/C specifications maintained by the 1394 Trade Association (1394ta) [1]. In AV/C, commands and responses are transferred between a controller and a target node in messages that are encapsulated within the FCP frame of an asynchronous firewire packet as shown in Fig. 2.

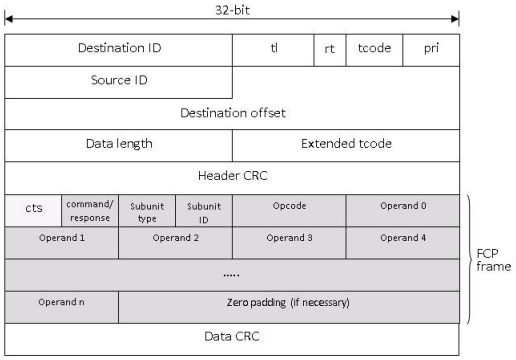

Fig. 2. AV/C command within the FCP frame of an asynchronous packet

The cts field is used to identify the command/transaction set utilizing FCP and has a value of 0x00 for the AV/C protocol. An AV/C message is either a command or a response. The AV/C general specification (2001)[4] defines various command and response values. Each command is targeted at a particular unit/subunit identified by the subunit type and subunit ID fields. The opcode is used to specify what operation to perform on the target using the operands as arguments for the action. The number of operands is dependent on the particular opcode.

An AV/C node is comprised of logical entities known as units. Within a unit are subunits that are functional entities that work together to achieve the overall role of an AV/C unit. The AV/C general specification defines various types of subunits. Within certain subunits such as the audio subunit are function blocks that perform specific processing on behalf of the subunit. Each unit has input and output plugs. A subunit has destination plugs that allow data into the subunit and source plugs that transmit data from the subunit. Function block input and output plugs also exists on a function block. An AV/C device possesses a maximum of 32 unit plugs and 32 subunit plugs [4]. There are different types of unit plugs, namely:

- asynchronous plugs - virtual end points of asynchronous data streams on an AV/C unit.
- isochronous stream plugs - virtual end points of isochronous data streams on an AV/C unit.

- external plugs - non-firewire plugs that transmit signal to and from a unit.

The AV/C protocol provides a suite of commands that can be used to determine the type of unit, subunits and plugs that exist within a device. These include the unit info command, subunit info command, and plug info command. Also defined in AV/C are connect commands, signal source commands, input signal commands and output signal commands used to determine or direct the routing of signals within an AV/C device.

An AV/C device presents a descriptor and its associated info blocks as an interface to any device or controller application on the bus that seeks to determine its internal details. Typically when a controller application discovers an AV/C device, it asks for its descriptors and info blocks in order to determine the internal nature of the AV/C device. After the controller application has determined the nature of the target, it then proceeds to send instructions (AV/C commands) or retrieve information about the state of various controls within the device. The nature of AV/C descriptors and info blocks are defined in the 1394TA documents titled '*AV/C Descriptor Mechanism Specification Version 1.0*' [3] and '*AV/C Information Block Types Specification Version 1.0*' [2], respectively.

However, AV/C defines several optional descriptors, which may be implemented by an AV/C device manufacturer. The availability of such alternatives makes it difficult for a controller node to parse with certainty, the various descriptors within any AV/C device, without prior knowledge of the descriptors implemented. Hence, a particular AV/C device - Terratec's Phase 24 FW - was used in this investigation, mostly because of its accessibility to the researcher.

B. AES-X170 Protocol

AES-X170 is a peer-to-peer command and control protocol that utilizes User Datagram Protocol (UDP) packets for the monitoring and control of networked audio and video devices [9]. The IPv4 (here after simply referred to as IP) provides a 32-bit addressing scheme for device identification on an AES-X170 network. Being an IP protocol, AES-X170 is independent of the transmission medium and has been used on firewire networks amongst '*IP over 1394*' enabled devices. The RFC 2734 (1999) document specifies how IP can be used over a firewire network [13].

Every AES-X170 device implements the AES-X170 stack [14]. It is the AES-X170 stack that creates AES-X170 messages for onward transmission and interprets received AES-X170 messages. This stack interacts directly with the IP stack on the device. Fig. 3 shows the layout of an AES-X170 device.

Every application that utilizes AES-X170 initializes the AES-X170 stack and creates an additional AES-X170 application node above the stack. It is within this AES-X170 application node that the functional elements that make up the device are modeled. The internal configuration node is created by the AES-X170 stack and holds general configuration and diagnostic information such as firmware version, IP address, the device type and the device name.

In AES-X170, a 7-level hierarchy is used to identify the functional elements and controls within a device. These elements and controls are referred to as parameters. A 'wild card' mechanism has been developed in AES-X170 that allows addressing of multiple parameters at any of the 7-levels. From an application developer's perspective, every AES-X170 parameter has associated with it a callback function that implements in a device specific manner the operation of the parameter. This layered hierarchy forms a tree structure with each of the 7-levels as a tree-node. The wild card can be used to address all tree nodes at any level of the hierarchy.

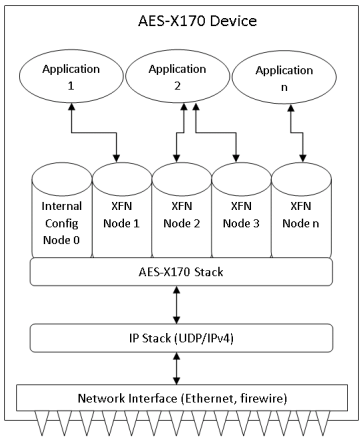

Fig. 3. Layout of an AES-X170 device (adapted from XFN Stack (2007))

An AES-X170 device's parameter can be addressed using either the full 7-level hierarchical address or the parameter's index. A parameter is assigned an index when it is being created by the AES-X170 stack. AES-X170 permits blocking and non-blocking transactions. In a blocking transaction a controller node sends a blocking request and waits for a response before proceeding with other operations. In a non-blocking transaction, the controller node sends a non-blocking command then proceeds to process other instructions. When it receives a response, it associates the response with the command it sent out using the sequence ID field of the AES-X170 message header. The layout of an AES-X170 message within UDP/IP is shown in Fig. 4.

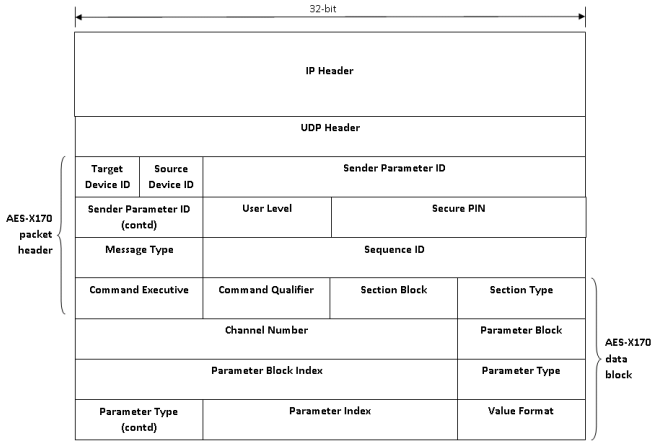

Fig. 4. Layout of a full data block AES-X170 message

AES-X170 messages indicate in their message header the identifier of both the target node and the source node. Security

and authentication are implemented with the use of the user level and secure PIN fields of the AES-X170 message header. An AES-X170 message can be either a request or response that uses either the 7-level hierarchy or parameter index to identify a parameter. The message type field indicates whether the message is a request or a response message, and whether the parameter is addressed using its 7-level hierarchy or using its parameter index. The sequence ID is used to reassemble packets at a receiving device. The command executive and command qualifier fields of the AES-X170 header are used to specify the exact instruction that should be carried out on the parameter [9].

The data block of an AES-X170 message can contain either the 7-level hierarchical parameter description or an index that uniquely identifies a parameter. Fig. 4 shows a full 7-level hierarchical description of an AES-X170 parameter. The 7-levels defined in AES-X170 are the section block, section type, channel number, parameter block, parameter block index, parameter type and parameter type index. A value format field that specifies the format of the parameter's value follows immediately after the parameter type index.

A conceptual understanding of AV/C and AES-X170 protocols form the foundation upon which the proxy is built. It is required that the proxy will understand both protocols in order to fulfill its role as a translator between devices on AV/C and AES-X170 networks. Proxies have been used in many applications, including web proxy, caching server proxy and content-filter web proxy. In certain cases, proxies have been implemented as firewalls to provide network security. This same advantage is obtainable when a proxy is used for integrating AV/C devices into an AES-X170 network. Another advantage to having a proxy that implements AV/C and AES-X170 protocols is that such a proxy is implemented at the (OSI/ISO) application layer. This means that the proxy performs a high-level abstraction of a device, and enables communication by sending protocol messages.

II. Design

The possible creation of a proxy as a means for audio/video network interoperability formed the basis of the implementation in this research. The conceptual idea was that the proxy will be situated in between various protocols from where it can receive messages from any of the protocols. On receiving a message the proxy interprets it and sends the appropriate instruction to the addressed device on a different protocol. To achieve this the proxy is required to understand all protocols it interacts with.

An AV/C proxy that understands both AV/C and AES-X170 protocols, was developed in this research. The AV/C proxy concept is shown in Fig. 5.

The AES-X170 proxy receives AES-X170 messages and fulfills them by executing the corresponding AV/C command.

The approach used to implement the AV/C proxy involved:

- acquiring an AV/C device, in this research Terratec's Phase 24 FW enhanced breakout box.
- determine the nature of the AV/C device (units, subunits and plugs) and its internal routing.

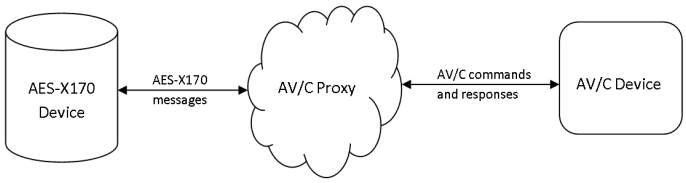

Fig. 5. Layout of AV/C proxy interaction

- model the AV/C device in terms of AES-X170 by creating AES-X170 parameters and implementing callbacks for parameters on the device.

The following use-case diagram shows the roles that are required of the AV/C proxy in terms of its interaction with AV/C and AES-X170 devices.

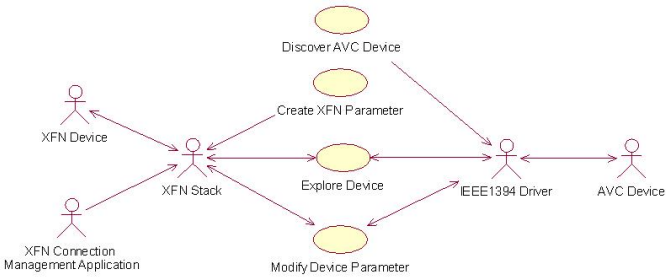

Fig. 6. Use-case diagram for AV/C proxy

III. Implementation

The current implementation of the AV/C proxy resides on the Ubuntu Linux platform. It uses the *raw1394* bus driver module to interact with firewire devices by utilizing the *libraw1394* application programming interface (API) [15]. This API enables an application to communicate with a firewire device by making calls to the raw1394 driver. The AV/C proxy also uses the *eth1394* module which provides for IP communication over the IEEE 1394 serial bus. Fig. 7 shows the interaction between the AV/C proxy and the low-level raw1394 drivers.

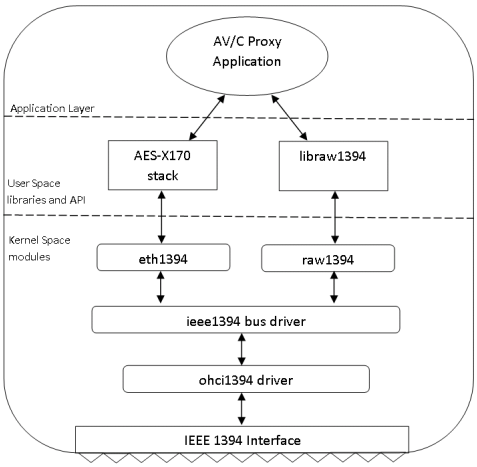

Fig. 7. AV/C proxy interacts with raw1394 and eth1394 driver modules

The *ieee1394* module mediates between the high-level kernel modules such as raw1394 and eth1394, and the low-level kernel modules in this case ohci1394. All interactions between the AV/C proxy and firewire devices are through the raw1394 module.

At start up the AV/C proxy gets a handle to the raw1394 driver, then it proceeds to discover all AV/C devices on the network. For each device discovered, the proxy creates an AV/C device object that models a physical AV/C device in terms of its globally unique identifier (GUID), unit and subunit types, the manufacturer identifier and number of isochronous stream input and output plugs. For each plug found an isochronous stream plug object is created. This plug object has the same attribute values as the physical isochronous stream plug it models. The plug attributes include the plug direction (input/destination or output/source), plug identifier and the transmission or reception channel set on the plug. Each AV/C object keeps a list of its input and output plugs. Fig. 8 shows the AV/C proxy class diagram.

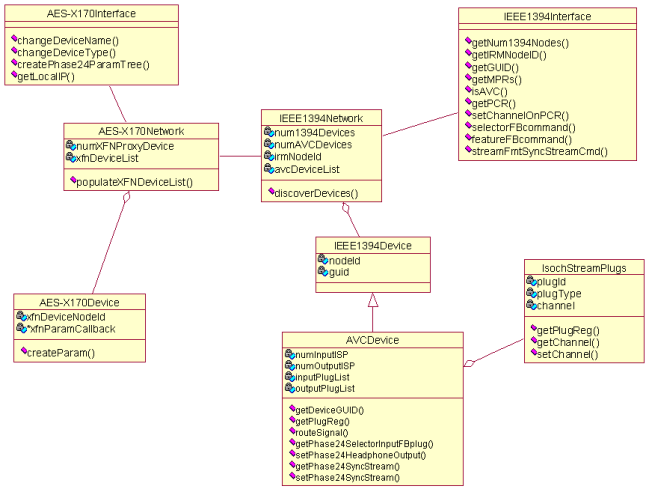

Fig. 8. Class diagram for AV/C proxy

In Fig. 8, the *IEEE1394Network* class holds a list of all AV/C devices. The AV/C proxy identifies a particular AV/C device based on its manufacturer identifier, device type and model textual descriptor. If the device is a recognized AV/C device, currently if it is a Phase 24 FW, the proxy then creates an AES-X170 proxy device object with its associated parameters to describe the recognized AV/C device. There exist a one to one mapping between an AES-X170 proxy device object and an AV/C device object using the device GUID. For each AES-X170 device object parameter created a corresponding callback is implemented to fulfill a request by triggering the appropriate AV/C command(s) addressed to an AV/C device.

The *AES-X170Network* class calls the *AES-X170Interface* class to initialize the AES-X170 stack. The *AES-X170Interface* class handles all interactions between the proxy and AES-X170 devices on the network.

IV. Testing and Results

The UMAN connection management application suite (UCMAN) is an AES-X170 connection management application. UCMAN was used to test the AV/C proxy for:

- discovery of AV/C devices on the network. Emphasis was on determining if UCMAN could identify the AV/C devices as AES-X170 devices.
- getting and setting the channel on an AV/C device input and output isochronous stream plugs.
- getting and setting connections between an AES-X170 device and an AV/C device's input and output multicores.
- performing internal signal routing within an AV/C device.
- getting and setting the sampling frequency on an AV/C device.

These form the basic operations performed on devices in a professional audio network. For all of the above tests, AES-X170 messages were sent to the proxy. It was the responsibility of the AV/C proxy to translate the received messages into AV/C commands.

Fig. 9 is a snapshot of the UCMAN device view when an AES-X170 device (UMAN evaluation board) and the AES-X170 proxy devices (Phase 24s) are discovered on the network. Each proxy device is identified by its AES-X170 node id in UCMAN. The AV/C proxy devices are the lighter coloured squares (purple squares) shown in Fig. 9.

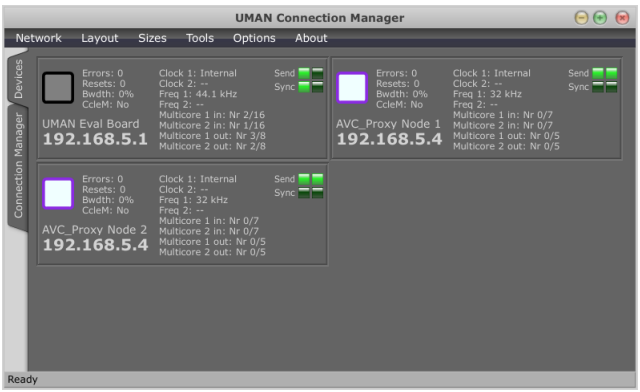

Fig. 9. AV/C proxy devices discovered by UCMAN

Fig. 10 shows the UCMAN network view. On the top-left of this view is the device matrix which shows all discovered AES-X170 devices. The top-middle is the multicore matrix that shows the multicores of two AES-X170 devices. The multicore view also shows the connections and current channels set on each multicore. The device on the left of the multicore matrix is the transmitting (source) device while the device at the top is the receiving (destination) device. The bottom-left of the Fig. shows the source device's internal routing matrix and the bottom-middle shows the destination device's internal routing matrix.

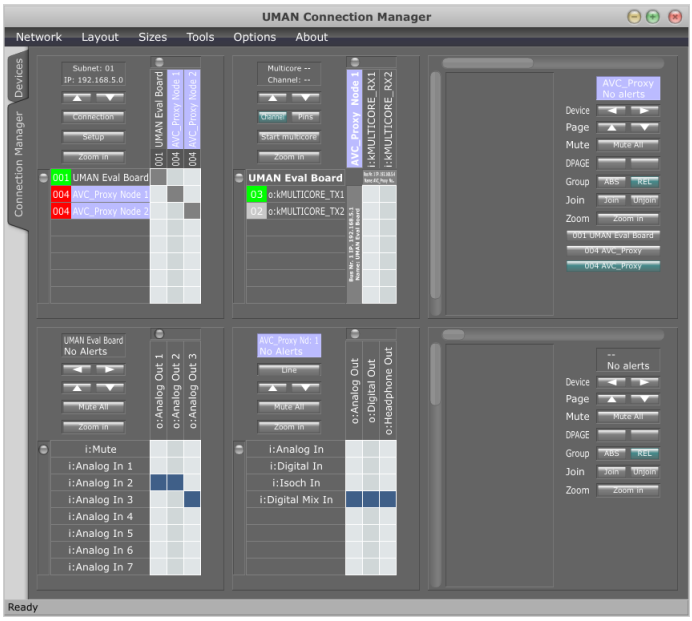

Fig. 10. UCMAN network view displays AV/C proxy devices

Colour coding is used to distinguish the AV/C proxy devices in UCMAN's network view.

V. Conclusions

Network interoperability can be achieved with the use of proxies. This research has demonstrated that devices implementing two different audio/video protocols namely, AES-X170 and AV/C, can communicate with each other by implementing an AV/C proxy. In this research, an application layer proxy was implemented that modeled an AV/C device in terms of AES-X170. Various tests were performed to affirm that connection management and internal routing was possible using the proxy.

The AV/C proxy created is an application level implementation that will allow for integration with other protocols. If such proxies were made commercially available, consumers will no longer be restricted in deploying any particular protocol for the networking of their devices. Similar proxies can be created between other audio networking protocols.

One added benefit is that by performing this high-level abstraction of devices in a proxy, a device conforming to one protocol can take advantage of features of another protocol. For example, although AV/C does not provide security and authentication features, the AV/C proxy makes it possible for an AV/C device to take advantage of these features as provided by AES-X170.

In the future, a framework will be developed that allows any AV/C device that implements the audio and music subunit to be easily integrated into modern audio networks with the aid of a single proxy.

References

[1] 1394 Trade Association. *1394 Technology - Specifications*. Available: http://www.1394ta.org/Technology/Specifications/specifications.htm [Accessed: 27 April, 2009].

[2] 1394 Trade Association. *AV/C Information Block Types Specification Version 1.0*, 2001.

[3] 1394 Trade Association. *TA Document 1999025: AV/C Descriptor Mechanism Specification Version 1.0*, 2001.

[4] 1394 Trade Association. *TA Document 2001012: AV/C Digital Interface Command Set General Specification Version 4.1*, 2001.

[5] 1394 Trade Association. "1394 TRADE ASSOCIATION PRESS RELEASE". 2007. Available: http://www.1394ta.org/press/TAPress/2007_1212.html. [Accessed 27 April 2009].

[6] Anderson D. *FireWire System Architecture*. Mindshare Inc., New Jersey, 2nd edition, 1999.

[7] Audio Engineering Society. *AES24-2006: AES standard for sound system control - Application protocol for controlling and monitoring audio devices via digital data networks - Part 1: Principles, formats, and basic procedures*, 2006.

[8] Cirrus Logic. "CobraNet CM-1 Digital Audio Network Interface Module", 2007. Available: http://www.cobranet.info. [Accessed: 27 April, 2009].

[9] Foss R. *XFN Protocol Overview*. Universal Media Access Networks, 2008.

[10] International Electrotechnical Commission - IEC. *ISO/IEC 13213: Information Technology - Microprocessor Systems - Control and Status Registers (CSR) Architecture for Microcomputer Buses*. IEC, 1994.

[11] International Electrotechnical Commission - IEC. *IEC 61883-1: Consumer Audio/Video Equipment - Digital Interface - Part 1: General* . IEC, 3rd edition, 2008.

[12] IEEE Std. 1394. *Standard for a High Performance Serial Bus*, 1995.

[13] Johansson P. *IPv4 over IEEE 1394*. Internet Engineering Task Force, 1999.

[14] Klinkradt B. *XFN Stack Overview*. Universal Media Access Networks, 2007.

[15] Linux 1394. *IEEE 1394 for Linux*. 2006. Available: http://www.linux1394.org. [Accessed: 27 April, 2009].

[16] Okai-Tettey H. *"High Speed End-To-End Connection Management in a Bridged IEEE 1394 Network of Professional Audio Devices"*. PhD thesis, Rhodes University, Grahamstown, 2005.

[17] Phillips J. "HiQnet Overview". Sound Marketing Central, 2005.

[18] Scavone G. *Open Sound Control (OSC)*. 2008. Available: http://www.music.mcgill.ca/ gary/306/week9/osc.html. [Accessed: 27 April, 2009].

[19] Wright M. *The Open Sound Control 1.0 Specification*. 2002. Available: http://opensoundcontrol.org/spec-1_0. [Accessed: 27 April, 2009].

OSEDUM P. IGUMBOR is currently studying towards a Masters degree in Computer Science at Rhodes University. Over the past 3 years he has been researching the integration of AV/C devices into contemporary audio/video network solutions.

RICHARD FOSS is an Associate Professor in the Computer Science Department at Rhodes University. His research interest is the networking of audio and video devices for live concerts, sound installations and recording studios. Over the past two years he has chaired various networked audio committees at the Audio Engineering Society (AES) conventions.