

QuickTime Streaming and IPv6

Daniel Grimm

*CTIE, Department of Electrical and Computer Systems Engineering
Monash University.*

daniel.grimm@eng.monash.edu.au

Abstract The Next Generation Internet (IPv6 based) is slowly being implemented and it is gaining further prominence with GPP support for 3rd generation mobile phones. One potential application of 3G phones and the available bandwidth is the use of streaming media (video and audio).

We believe that, to become widely available these phones would need to be able to access the existing content on the IPv4 Internet rather than just dedicated IPv6 streaming servers. To Address this issue, we have initiated a project within the Applications Program of ATcrc (Australian Telecommunications Cooperative Research Centre) with the support of an AUDF Developer Grant, and converted the IPv4 based Darwin Streaming Proxy to IPv6. The proxy provides an application layer gateway between IPv6 streaming (RTSP/RTP) clients (such as MPEG-4 or (modified) QuickTime players) and IPv4 based streaming servers that already exist on the Internet eg., QuickTime/Darwin Streaming Servers (DSS). Students including B.E (ECSE) vacation experience project, 4th year thesis and B.Tech (Computer Studies) Industrial Design project groups have been used to implement proof of concept clients for IPv6. The porting of the Darwin Streaming Server to natively support IPv6 is an ongoing project.

Introduction

IPv6 has been proposed as the supported protocol for third generation (3G) mobile phone systems (3GPP UMTS and IMS[1, 2]) and this poses IPv4-IPv6 transition issues for existing classes of services (web, streaming media, Voice over IP (VOIP)) on 3G capable devices. One approach to streaming media to IPv6 capable clients is to use native services - which means updating the servers to directly support IPv6. Current QuickTime/Darwin Streaming Server (DSS)[3] has no IPv6 support, similarly for most of the installed Microsoft Windows Media[4] or RealNetworks Servers[5]. The AUDF Developer Grant project was to extend DSS so that it supported IPv6 streaming. The project is still ongoing.

It is presumed that there will only be a gradual uptake of IPv6 supported server applications, meaning that large amounts of content will not be directly available to the new devices.

An application layer gateway (ALG) for streaming media is one solution to address the problem. This paper describes one implementation of a non-caching proxy that performs the function of an IPv4 to IPv6 ALG for streaming QuickTime/MPEG4 (RTSP/RTP) media using TCP and UDP. Figure 1 illustrates our approach.

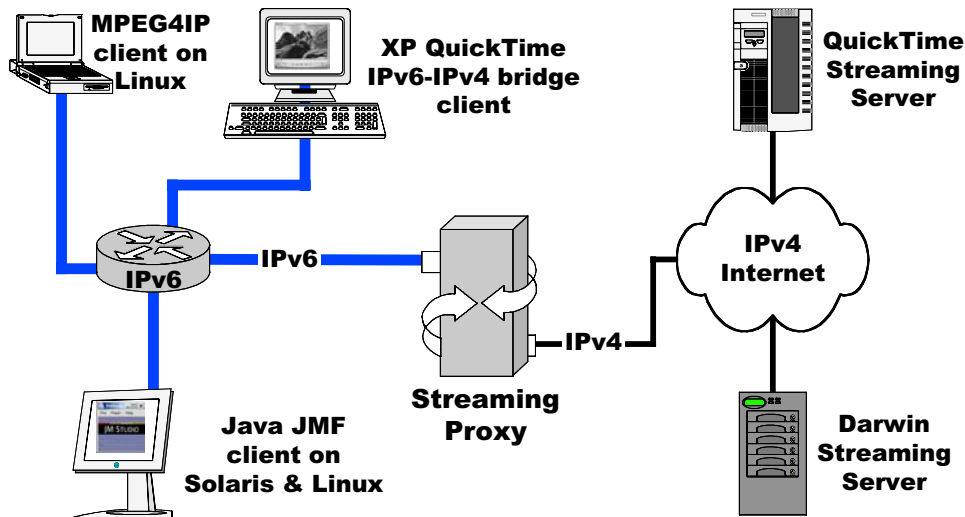


Figure 1: IPv6 Streaming Proxy linking IPv4 servers on Internet to IPv6 network and clients

The proxy translates the text-based Real Time Streaming Protocol (RTSP[6]), which is transported over TCP. The proxy rewrites IP addresses and port information as appropriate and updates the SDP[7] content of the RTSP messages to meet IPv6 requirements. The proxy also translates the Real Time Protocol (RTP[8]) content from IPv4 RTP/UDP packets to IPv6 RTP/UDP packets.

The proxy has been demonstrated to work with IPv6 implementations of the MPEG4IP[9] Linux based QuickTime/ISMA compliant MPEG-4 player, and with a Java JMF[9] media player. A Windows based IPv6 QuickTime player is under development at present as an undergraduate thesis student project.

QuickTime Streaming Technology

QuickTime Streaming is seen to be particularly suitable for IPv6 research due to its applicability to future networks such as the 3G cellular systems. MPEG-4 Streaming (ISMA[10]) is based on the QuickTime file format, and uses the same RTSP/RTP protocols.

Streaming QuickTime uses a URL to request a video/audio stream. For example, to request a video stream called sample.mov from the server 'snowy.ctie.monash.edu.au' a URL of the form `rtsp://snowy.ctie.monash.edu.au/sample.mov` is used. The URL itself does not specify which version of IP (IPv4 or IPv6) is to be used.

The standards based RTSP protocol is used by the player (and server) for signalling. RTSP protocol covers signalling areas such as requesting the details of the movie codecs required, number of tracks, source IP addresses and ports of the server sending the stream.

RTSP messages also include the usual player type commands like play, pause, start at time 'nnn'. RTSP players and servers can negotiate on some parameters related to the requested stream. For operational details of RTSP, the reader can refer to the specification document RFC2326[6] as it has a very readable description of how RTSP is to be used.

Actual movie content is transported by the RTP protocol.

RTP over UDP is used to transport the encoded video or audio tracks (or other types of tracks). Again it is worth noting that the RTP protocol does not specify the IP layer (IPv4 or IPv6) required to transport it. RTP uses two sequential ports for each track of the stream. One port is used to carry content (say a video track) and the second (the RTCP port) is mainly used as a feedback mechanism to report on quality and packet loss. Instant-On QuickTime streaming uses this RTCP feedback mechanism amongst others.

From a network programming point of view it is worth noting that RTSP over TCP means that there is no loss of signalling packets (losses in the network lead to resends of data), but audio and video content can be lost as the RTP packets are transported over UDP. UDP is a protocol that has no inbuilt mechanism for resend of missed packets.

For my work I chose to ignore the use of RTSP/RTP over HTTP. Partially this was due to lack of clients, but also due to the inherent extra bandwidth consumed by the HTTP protocol. It was not seen as worth pursuing as it increases the bandwidth requirements of the streaming video without increased reliability.

For mobile devices on a 3G network, bandwidth minimisation is important, as telephone network operators usually charge on a per byte basis. Using HTTP does have some convenience when dealing with firewalls, which is why the streaming server supports it.

The DSS server software is part of the Apple Open Source software initiative, which has led to numerous groups choosing to use it as a platform for networked streaming video research. The proxy provided with the DSS server is very limited and was the basis for the IPv6 proxy.

Mac OS X IPv6 support and development environment

The original intent of the developer grant project had been to use Mac's for all work. Unfortunately that was overly ambitious, as the OS X 10.1.x system did not have sufficient support for IPv6 work. The IPv6 stack within the operating system was based on a very old Kame[11] port, and there were no IPv6 developer tools. Time was spent updating the kernel to support IPv6, before realising that the developer tools did not have the libraries required for IPv6 program development. There were workarounds available by installing part of the Kame developer libraries and source, but eventually it was decided to do the development and testing work on Linux instead due to its up to date IPv6 stack and developer tools.

Mac OS X 10.2 Jaguar brought good IPv6 support and developer tools. Unfortunately it came out too late for my use, although I did investigate the beta version.

For the IPv6 Streaming Proxy a class library for IPv6 networking called SocketCC[12] was used and this now has Mac OS X 10.2 Jaguar support. This means the proxy is usable on Mac OS X as well as Linux and Solaris systems.

Client Development: QuickTime API - NO IPv6 support, No access to RTSP messages.

The QuickTime API (QuickTime 6 SDK) allows applications to be built that support QuickTime. There is a whole section of the API to support streaming media, and it was believed that a client application could be built that would support IPv6. Unfortunately, the RTSP protocol is not exposed by the API, so modification to support IPv6 addresses in the

RTSP messages was unable to be done. Instead a client was created that generated its own IPv6 RTSP messages and bypassed the QuickTime API for RTSP. See below.

IPv4 - IPv6 transition programming issues

The DSS and DSS-proxy code had been written in a typical IPv4 specific way — using Integer or 32bit numbers to represent the IP address. Manipulation of addresses by associating or copying binary values over integer values and relying on bits being in the correct order were used. This is considered to be poor programming technique now that IPv6 is becoming common. The most obvious reason for this is that IPv6 addresses require 128 bits.

It was decided to use a derivative of the IPAddress class from the SocketCC library to replace the integer IP addresses used throughout DSS source code. This turned out to be a much larger task than expected, with large chunks of code that I had not investigated when submitting the developer grant needing changes. Some parts of the code are so heavily dependent on integer representation of the IP address that they require a complete rewrite (eg: socket pool hashing function used the binary representation of the IP address integer as its hash value). Due to the complexity of some of this code, the porting to IPv6 of the DSS server was delayed, and extra effort was put into updating the proxy code.

The RTSP[6]specification supports IPv6 URL's and IP numbers, but does not explicitly describe how to represent them. The programming was done based on the descriptions in the RFC for SDP[7], IPv6 support for SDP[13] and the ongoing work towards SDPng. In addition, text representation of IPv6 addresses followed the appropriate IPv6 URL addressing RFC[14]. Eg: 2001:0db8::12:12bc is a Global scope IPv6 address.

One major issue showed up, in that it is difficult to program for, and ensure, that global IPv6 addresses are used for both text representation and UDP/IPv6 packet addressing (in the actual IP on Ethernet frame). As RTSP is text based, it is essential that a consistent global IPv6 address is used for both the source IP in the RTSP Setup response messages and also in the actual RTP/UDP/IPv6 streams.

The SocketCC library was updated during this project to ensure that the correct addresses (either as ASCII strings or within TCP or UDP packets) are used.

QuickTime/Darwin Streaming Server (DSS) Porting to IPv6

The AUDF grant request had as its main aim to port the DSS code to natively support IPv6. The grant was put in based on a quick look at the source code, identifying six main socket related files that would need modifying. The DSS source repository is large (approximately 70,000 lines of code). As it turned out, although the DSS source is portable (can be compiled on many systems), modifying the socket related files was not sufficient. The expected propagation of small changes into other files turned into large changes in lots of files. This was mainly due to the use of 32bit integers to represent IP addresses.

Changing from an integer to an IPAddress class object led to the discovery of many initialisation and parameter passing issues that are still to be fully resolved. Similarly the differences between IPv4 and IPv6 multicast programming have to be implemented. Due to the timing of this project (pre-Jaguar) the bulk of the work was carried out on Linux systems.

There has been interest in the project from all around the world — mainly in University and IPv6 research consortiums. One consortium (6Net.org) has an IPv6 native DSS ‘hack’ (to use their terminology) based on a different set of network libraries.

Darwin Streaming Proxy - port to IPv6

Requirements of a Streaming Proxy

To act as an Application Layer Gateway to provide RTSP and RTP Streaming media to an IPv6 client, a Streaming Proxy needs to translate the RTSP protocol and copy RTP packets from UDP on IPv4 to UDP on IPv6.

The RTSP protocol consists of a set of relatively straightforward text (readable) messages. Of primary interest to a proxy are the DESCRIBE and SETUP messages and their OK responses from the server.

The DESCRIBE message response contains the IP address of the owner of the media stream, and the track information, including who to send control messages to. For streaming video on demand from a QuickTime/Darwin Streaming Server the owner IP address is the IP address of the server contained in the request URL. This remains unchanged in the proxy.

The SETUP messages contain the source IP address of the RTP stream for each track — the proxy needs to replace this IPv4 address string with its own IPv6 global address.

For example:

```
RTSP SETUP OK message
Transport:client_port=6970;server_port=10000;source_ip=10.1.252.63
Changes to
Transport:
client_port=6970;server_port=10000;source_ip=2001:0db8::12:fe80:ab12
```

The streaming video and audio RTP on UDP streams need to be received by the proxy and then forwarded to the IPv6 client in IPv6 UDP packets. There is no need to alter the RTP payload, so effectively the proxy receives UDP packets, reads the payload into a buffer and forwards it out to an IPv6 UDP socket.

Streaming Proxy Implementation

The Improved Streaming Proxy was based on Apple's old C proxy code — basically a polled, shared socket, RTSP and RTP proxy program written for Unix systems with IPv4 sockets.

This structure of the program was unchanged, leaving the logic intact. Only code related to TCP and UDP sockets was altered to support IPv6. Initially the code was rewritten in classic BSD socket programming fashion with correct type casting of IPv4 addresses (instead of using integers to hold IP addresses). The code was then rewritten using the class library called SocketCC for ease of programming and potential IPv4/IPv6 transparency.

The RTSP specification allows extensions and options that are not supported by the implemented proxy.

Streaming Proxy Experimental Work

The Improved Streaming Proxy has been compiled on Linux, MacOSX (10.1 and Jaguar 10.2) and should also compile on Solaris as the SocketCC library it is based on has been ported to all systems.

Running as a user process on a Pentium II Linux box it is easily able to cope with 4 clients (The code has been compiled with a limit of fifty clients) viewing videos with bandwidths ranging from 28kbs to 700kbs simultaneously. The clients were initially tested using the proxy as an IPv4-IPv4 proxy, but also as an IPv4-IPv6 application layer gateway.

The improved proxy shows no packet loss when a client calls a DSS server that has a feature called 'reliable UDP' turned on — the original Apple code causes a 40% packet loss at the client due to incorrect handling of the RTP/RTCP packets soon after the start of streaming.

The MPEG4IP Linux streaming MPEG4 player had no support for RTSP proxies, so it was modified to support the use of a proxy. It was then used to trial the full IPv6 RTSP/RTP streaming video solution via the improved proxy. Aside from adding IPv6 proxy support, no other changes were required to be made to the MPEG4IP programs — verifying compatibility between my IPv6 RTSP messages and another implementation. The MPEG4IP player is capable of playing industry standard ISMA streaming MPEG4 (which can be served by a DSS server amongst other systems).

Fragmentation of the IPv6 packets containing the Video RTP packets was observed due to the increased header size of the IPv6 UDP packets.

Various student and vacation worker projects have been underway to provide client applications that support IPv6. Java 1.4 supports IPv6, but the JMF (java media framework) has not been written sufficiently transparently to IP type to allow it to be directly used to play IPv6 based streaming video. A current B.Tech. student group project is seeking to implement a JMF IPv6 player. Apple currently has no IPv6 support in its QuickTime players, and a vacation work project currently under way is demonstrating a proof of concept method to use the QuickTime API on Microsoft's XP platform to play streaming video over IPv6.

DSS 4.1.2+ Server and QuickTime 6 issues

The streaming proxy was updated to support some features used by DSS4.10 servers (including having track sources coming from different IP numbers) while QuickTime 5 was current. Since then we have progressed to QuickTime 6.1+ and extra features like Instant On have been added. The DSS server already had support for these features, but the proxy does not. The extra RTSP messages required by Instant On (identified by an x-dynamic-rate=1 in the RTSP Setup message) caused the streaming proxy to ignore the connection as it is pre-programmed for certain RTSP state transitions and sequences. The problem was resolved within the proxy, by setting the x-dynamic-rate to 0, and rewriting the User-Agent field string. The changes make the client requests appear to come from a non-QuickTime client that does not support Instant On.

QuickTime (RTSP/RTP) IPv6 CLIENT programs

MPEG4IP for Linux — MPEG-4 player compatible with QuickTime streams

The MPEG4IP project on sourceforge.net provides a QuickTime compatible MPEG-4 player. It is a project that was founded by some CISCO workers, bringing together a large collection of available code libraries for Linux systems. Originally they distributed the Darwin Streaming server version 2 with their collection of code. The 6Net.org consortium's applications project workers had added IPv6 functionality to the MPEG4IP player, and this has been used to test and demonstrate IPv6 based MPEG-4 video streaming via the proxy. The code to support a proxy was added by myself into the RTSP communications library of the MPEG4IP player.

IPv6 and the Windows QuickTime SDK

An ATcrc Vacation Scholarship worker was given the task to identify and build an IPv6 QuickTime player application for windows. Unfortunately the QuickTime SDK does not reveal the RTSP protocol, so IPv6 is unable to be added using the SDK. A partial client was written with the RTSP protocol and user interface being coded explicitly to support IPv6. The RTP stream was then received from IPv6 and passed on to an instance of the QuickTime player. The player opened an SDP file (containing configuration information for the streams) generated by the application. This combined application proved to be rather inefficient, but led to the development of the much more capable client described next.

Local Bridge IPv6-IPv4 client for Windows XP QuickTime Player

This application was created under ATcrc Vacation Scholarship and provides a bridge between QuickTime and the IPv6 streaming world, using a local proxy (within the windows workstation) that performs RTSP proxying and RTP bridging functions. A final year bachelor of engineering student has been further developing the code as his thesis project. The application requires no modification to the Apple distributed QuickTime player, and does not use the QuickTime SDK. Instead it creates IPv4 and IPv6 sockets for both RTSP and RTP and passes the incoming RTP and RTSP messages to the QuickTime player. The RTSP messages are rewritten to provide a unique User-Agent string, and support IPv6 when communicating with a server or streaming proxy. Incoming RTSP messages are converted where appropriated so that the standard IPv4 only QuickTime player can use them. The QuickTime player only needs to have its streaming proxy preference configured to use the localhost IP address and Port of this bridge application for it to all work. At present the windows local bridge/proxy consumes too much CPU time due to the polling method implemented, and is undergoing a rewrite.

Java JMF client

The Java Media Framework (JMF) has RTSP/RTP support and can communicate with a QuickTime streaming server. JMF has limited codec support, so specially made QuickTime movies were used for the testing. JMF has an MPEG-4 codec provided by IBM, but during the research period, IBM withdrew that codec due to licensing issues (which have now been resolved).

JMF was investigated using both an ATcrc Vacation Scholarship worker and B.Tech (Computer Studies) Industrial Design project group of students. Initial work showed that Java v1.4 and JMF could support and receive IPv6 RTP streams, but further work was needed to get the RTSP protocol supporting IPv6. This work is ongoing by the student group.

Conclusions and Future Work

The AUDF grant application that I had submitted turned out to be too ambitious but provided a good framework for investigating IPv6 based streaming media. The AUC website's visibility on the web meant that there was a relatively constant stream of enquiries for IPv6 enabled QuickTime streaming server code from overseas. Some collaboration has occurred with a number of these overseas researchers.

The Improved Streaming Proxy provides a relatively straightforward way of testing IPv6 QuickTime streaming. It correctly handles basic RTSP/RTP streaming media, but does not handle all of the extended RTSP messages used by the latest QuickTime players. The proxy is sufficient to demonstrate streaming MPEG-4 (ISMA[10] standard) in addition to other codecs suitable for playing via Apple's QuickTime player.

The code is not particularly robust as it was assembled as a proof of concept, rather than an end product.

The experimental work has shown IPv6 fragmentation of the RTP payload and this leads to further work and investigation into the potential effects of fragmentation and repacketisation. Re-hinting the video file to use smaller maximum packet size reduces the risk of fragmentation.

The AUDF Developer Grant and Apple's Developer Connection programme and Open Source initiative have been extremely helpful in providing a framework for exploring IPv6 issues related to streaming media.

Acknowledgements

The author would like to thank the AUC for the AUDF Developer Grant that helped support this project, which was carried out as part of the Australian Telecommunications Cooperative Research Centre's Applications Program.

References

- [1] WASSERMAN M. (2002) *RFC 3314: Recommendations for IPv6 in Third Generation Partnership Project (3GPP) Standards* IETF.
- [2] 3GPP (2002) *3GPP TS 23.228 v 5.3.0, IP Multimedia Subsystem (IMS); Stage 2 (Release 5)*.
- [3] APPLE COMPUTER INC. (2002) *Darwin Streaming Server* <http://developer.apple.com/darwin/projects/streaming/> .
- [4] MICROSOFT INC. (2003) *Windows Media Server Services* <http://www.microsoft.com/windows/windowsmedia>.
- [5] REALNETWORKS (2003) *Helix Servers* http://www.realnetworks.com/products/media_delivery.html.

- [6] SCHULZRINNE H., RAO A., ET AL. (1998) *RFC 2326: Real Time Streaming Protocol*.
- [7] HANDLEY M., AND JACOBSON V. (1998) *RFC 2327: SDP Session Description Protocol*.
- [8] SCHULZRINNE H., CASNER S., ET AL. (1996) *RFC 1889: RTP: A Transport Protocol for Real-Time Applications*.
- [9] SUN MICROSYSTEMS INC. (2003) *Java Media Framework API*
<http://java.sun.com/products/java-media/jmf/index.html>.
- [10] ISMA (2003) *Internet Streaming Media Alliance* vol. 2003: www.isma.tv.
- [11] KAME PROJECT (2002) *Free IPv6 stack for BSD variants* <http://www.kame.net/>,
www.kame.net.
- [12] CTIE (2002) *SocketCC — a C++ socket class library*
<http://www.ctie.monash.edu.au/socketcc/> Monash University.
- [13] OLSON S., CAMARILLO G., ET AL. (2002) *RFC 3266: Support for IPv6 in Session Description Protocol (SDP)*.
- [14] HINDEN R., CARPENTER B., ET AL. (1999) *RFC 2732: Format for Literal IPv6 Addresses in URL's*.