Embedded Streaming Media Servers

Class #350

Mike Ficco & Brian Jupin

Embedded Systems Conference
San Francisco, 2004

Introduction

Internet based Streaming Media Servers have been around for a few years now. In general, they provide the ability to deliver a relative low bit rate media stream through the Internet to a client running on a PC, workstation, or other device. Embedded Streaming Media Servers (ESMS) are a little bit different animal. Their intended usage is significantly different and the entire concept is much less mature than its Internet oriented cousin. It may even be true that the ultimate ESMS usage doesn't exist yet and must still be visualized in the (perhaps) distant future. What a wonderful opportunity and challenge for embedded designers. Today they can design products to fill the immediate needs. They also have the opportunity to define the capabilities and operation of future generations of in-home media servers. For the embedded systems designer, Embedded Streaming Media Servers present the frontier of sophisticated mass-market consumer equipment. Inexpensive extremely large hard drives and the ever falling price of memory and high-powered processors make consumer media servers economically feasible. This paper discusses various aspects of building Embedded Streaming Media Servers. It assumes some familiarity with embedded systems.

Internet Vs. Embedded: The Server

Internet Media Servers may generally be expected to consist of sophisticated software running on high-powered platforms in relatively large corporate installations. They may be simultaneously accessed by numerous programmers and be running multiple applications in addition to serving a large number of simultaneous media streams. To provide all this needed functionality, they likely have robust security, run a variety of applications, and offer monitoring, diagnostic, and debug tools and logs. They must handle real-world Internet issues such as load balancing, denial-of-service attacks, viruses, worms, and hackers. In effect, they operate in the worldwide Internet community and must cope with the worst that it has to offer.

In a "settop appliance" incarnation, an Embedded Streaming Media Server may sit in someone's house and serve a movie to the bedroom, a movie to the family room, and music to the den. It could operate in a closed environment or perhaps in a proprietary network and therefore have much less need of protection from the evils of the Internet world community. A slightly more complex ESMS would allow the owner to remotely communicate commands and retrieve information from the settop. However, ESMS designs envisioned by this paper are not intended to be general purpose Internet media servers. In this paper's view, a settop Embedded Streaming Media Server may be much more versatile than its Internet cousin in the way it acquires or receives media to be streamed. The ESMS may be required to digitally record live programming for immediate or future distribution. It is this Digital Video Recording (DVR) functionality combined with a program guide that dramatically distinguishes the usage of ESMS from typical Internet servers. A settop appliance with DVR, a program guide, and the ability to stream media brings us ever closer to the holy grail of "any move, any time".

Internet Vs. Embedded: The Client

Internet Media Clients are available from a variety of manufacturers and run on several platforms and operating systems. As a group, they are sophisticated programs that offer pluggable CODEC modules, advanced user interfaces, and a standard TCP/IP connection to the Internet. Detailed discussions of Internet media clients are beyond the scope of this paper. However, some consideration of these is necessary to insure an achievable and compatible feature set is defined for the ESMS. The cost and complexity of Internet servers relegates them to industrial applications. Internet clients, however, are already in ubiquitous household use. It is, therefore, an honorable and achievable ESMS goal to be serving a movie to an Internet client running on a PC or MAC in the den while simultaneously serving a movie to an embedded client showing the movie on a TV in the family room. In this scenario, the embedded media client may actually be an economical settop cost optimized for this purpose. Finally, essentially the same embedded media client may run within the ESMS itself to present a movie on a connected television.

Reference Feature Set: The Server

In any project, the requirements should be the first thing established. In our case this corresponds to the feature set supported by the Embedded Streaming Media Server discussed in this paper:

- The ESMS must be able to acquire and store content. This may involve the real-time ("live") capture of a digitally encoded transmission, digitizing and compressing of analog programs, or some form of background (non-real-time) acquisition. The server should be able to acquire multiple different programs simultaneously.
- The ESMS should have the ability to simultaneously serve multiple streams of digital programming (multimedia) to network connections.
- The ESMS may have the ability to play (output) analog programming to a TV via NTSC/PAL, component video, s-video, etc. This output may include picture-in-picture of a different program.
- The ESMS should be able to perform all media acquisition and serving functions at the same time. It would be acceptable that the sum of the number of acquisition and serving activities be limited to a maximum number. That is, one may choose more servings at the expense of fewer simultaneous acquisitions.
- The ESMS must be able to handle serving heterogeneous media. That is, the server must be able to deliver both movies (video and audio) and music (audio only).
- The ESMS must be able to display a list of available programs and should allow the user to navigate the list and select a program to watch. List display and navigation may be local via a connected TV screen or remote via a network connection to an embedded or Internet client.
- The ESMS should provide the ability to delete, hide, and otherwise manage the recorded programs.
- As suited to the current state of the program, the ESMS should be able to perform the normal functions we've come to expect from VCRs (pause, play, fast forward, rewind, etc.).
- The ESMS should have the ability to perform advanced functions only possible with hard disk based digital recording. These may include, for example, the ability to instantly reposition to another part of a playing program, to toggle viewing between two programs without missing anything, dynamically insert personalized information into a playing program, or perhaps to video edit recorded programs.
- The ESMS should be able to receive and process a PlayList of programs.
- The ESMS must provide point-to-point (TCP) connections to clients.
- The ESMS must be capable of acting as an HTTP server to stream media to Internet clients.
- The ESMS should be capable of multicast (UDP) broadcast of media

Reference Feature Set: The Client

Following are desirable features of a client compatible with our Embedded Streaming Media Server. Note that this feature set is applicable to both Internet and embedded media clients. Today's Internet media clients generally do not excel at supporting some of these features. They are more suited to real-time delivery of relatively low bandwidth data (such as the Internet web cast of a conference). This is part of the joy of helping to define the capabilities of future product generations and offers manufacturers the opportunity of distinguishing themselves with a custom client:

- The client should be able to initiate an HTTP stream from a specified URL.
- The client must be able to achieve audio/video synchronization from possibly widely skewed media.

- The client should be able to tolerate wrapping of the media timestamps
- The client should be able to tolerate gaps in the transmission of the audio and/or video and should be capable of promptly achieving audio/video synchronization after such gaps.
- The client must be able to display a list of previously recorded media
- The client must be able to display a program guide of currently broadcast media
- The client must be able to pause the display of the media
- The client should be able to fast forward and rewind the streaming media.
- The client should be able to handle PlayList of programs to enable hours of preprogrammed viewing.

Embedded Platform Architecture

Overview

This paper focuses on the embedded components and modules involved in building a settop appliance Embedded Streaming Media Server. Such a settop appliance is assumed to operate in a digital TV network. Hence, it is assumed that such critical modules as the power supply, tuners, program guide, front panel, infrared receiver, etc. simply exist and work. Obviously a great deal of variation is possible in the design and implementation of these modules. We assume that such variation is constrained to remain compatible with the goal of acquiring and streaming media to clients in a typical household.

Major Modules

The emphasis of this paper is the streaming distribution of media. As such, it is not concerned with modules controlling generic operation (such as managing the front panel) and only minimally concerned with modules involved in the acquisition of media. It is sufficient to note that the acquisition of media is by design and intent segregated from the serving. That is, streaming or playing the media has no effect on any planned or in progress acquisition other than the necessity of sharing system resources such as memory, processor cycles, and hard drive throughput. It is expected that acquisition of media will continue without material interruption while various clients start and stop presentation of media streams.

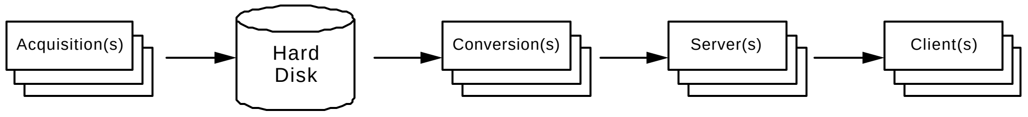

Figure 1. Embedded Streaming Media Server Major Modules

Acquisition

The *Acquisition* modules are the source of all multimedia data received and stored by the Embedded Streaming Media Server. Multiple acquisitions can be in progress at the same time. Some may be live and real-time while others may be trickling in at a very slow rate or blasting in much faster than real-time. While details are beyond the scope of this paper, it is reasonable that a distinct *Acquisition* module be associated with each type and source of data. Each *Acquisition* module would uniquely understand details of the input being received. For example, one type of *Acquisition* module may understand a MPEG

Transport Stream sent to cable or satellite settop boxes. Transport Streams are generally designed to be able to tolerate some amount of data loss in transmission and an associated *Acquisition* module would understand the details and consequences of such a transmission. Other inputs may expect or insure full data integrity. In a complex service network, a wide variety of *Acquisition* modules may exist to handle an assortment of compression, encoding, or encryption schemes. The intent of all *Acquisition* modules is to remove any proprietary formatting, encoding, or encryption and record all received media in a common (i.e. native to this settop design) format. This operation may involve significant processing or reformatting but once completed all multimedia data is stored in a common format. Note that this does not mean that all the recorded data is of the same type. The ESMS is designed to support the acquisition, storage, and streaming of heterogeneous multimedia data. The same server may store and serve PCM audio, MP3 audio, MPEG 2 video, and proprietary compression and encoding schemes. Each file is tagged with sufficient metadata to allow a *Format Conversion* output module to rearrange the content as necessary to be compatible with the destination client. The metadata could also contain other program related information such as the title, plot summary, time of recording, etc.

It is worth noting that the actual media record operation could be segregated from acquisition. An elegant *Acquisition* design could simply tag the "common format" data with the associated destination file and place it on a *Record Queue* service by a generic *Record Thread*. A high-end system may have more than one hard drive, but it is reasonable that there is a single *Record Queue* and associated *Record Thread* for each hard drive in the settop. This is because a physical hard drive can only be reading or writing a single location at any given time. Hence, all accesses of a hard drive must be synchronized. Synchronizing reads and writes must be done with a protective mutex, but using a single write thread for each hard drive naturally forces sequential processing of queued hard drive write requests and automatically results in serialized disk access. Actual writing to the hard drive is best accomplished using Direct Memory Access (DMA). DMA speeds the transfer and relieves the processor from the labor of being directly involved in moving the data.

Storage

Today the storage module of a consumer priced Embedded Streaming Media Server will typically be a large capacity economical hard drive. Although the focus of this paper is on serving the stored media, it is worth noting that a great variety of information and data will likely be stored on the drive in addition to the media. Adding a hard drive to a settop not only adds the cost of the hard drive itself, but also likely increases the cost of several other components of the product. To accommodate the hard drive, the box must become bigger, the power supply must become heftier, and additional heat will be generated. Cost conscious engineers will be looking to use the hard drive as much as possible to replace storage previously needed in the basic settop design. Certainly it is reasonable to expect the hard drive to hold the executable code, debug and log files, program guide information, and perhaps to replace large RAM buffers in the transmission and reception of commands and data on a LAN or modem connection.

It is, therefore, likely that the ESMS hard drive will have at least two partitions and perhaps two completely different file systems. It is probably best that the partition that holds the executable program and log files be the standard file system provided by the operating system of choice. The multimedia partition, however, may benefit greatly from a custom file system. Desirable attributes of this custom multimedia file system include:

- The file system must allow multiple open instances of each file. Certainly several clients could be receiving the same file (but perhaps each from a different position in the file). In the case of viewing a live program, the file would be open for both reading and writing. The maximum number of open read instances is a fundamental design issue and depends on the processing power and other resources

of the platform. For the ESMS application, it is reasonable that only a single record instance be allowed for each file.
- The file system must support heterogeneous data storage. That is, the file system must be able to store audio only and audio/video programs. It is also reasonable that the file system not restrict the types of compression and encoding used by the media.
- The file system should store programs using large contiguous disk blocks. By far the greatest delay in accessing a hard drive is moving the drive head to the correct position. Efficient use of the hard disk requires that data be read in large contiguous blocks to minimize head motion. The optimum size of the block depends on many factors and should be carefully analyzed early in the development cycle.
- The file system must support very large files. A single long movie (say three hours) recorded at 10 megabits per second may require up to 14 gigabytes of disk space. Depending upon the intended market of the product, the files system may reasonably be expected to handle individual files of perhaps 64 gigabytes.
- The file system must support deletion of files in a fashion that minimizes or eliminates disk fragmentation.
- The file system should support efficient video editing by allowing removal and insertion of file sections without forcing the potentially huge amount of data in the file to be copied to a new file.
- The file system should support hiding files and protecting files from unauthorized deletion.

The actual format in which the multimedia data is stored is another fundamental platform design decision. Some current MPEG based digital recorder designs use essentially the original MPEG Transport Streams, some use formats roughly approximating MPEG Packetized Elementary Streams, and some use highly proprietary configurations. Each approach has advantages and disadvantages that must be considered early in the platform design. Furthermore, initial requirements must state whether the media stored on the hard drive must be encrypted. If encrypted, additional hardware or processing power is needed to encrypt and decrypt. A determination must be made as to whether encryption must be real-time and whether the server or the client itself will perform decryption. Finally, the designers must decide how to generate encryption/decryption keys and how to store and distribute them. The earlier these decisions are made in the design process the better since retrofitting encryption, decryption, and key distribution into an existing design can be very difficult.

Format Conversion

It is reasonable that a *Format Conversion* task or thread be assigned to each server connection. Format conversion should be viewed as the method that builds the media stream into a format understood by the destination client. Format conversion should be distinguished from the transmission protocol (such as HTTP). The transmission protocol is handled by a *Server* thread itself. Examples of format conversion include decrypting the data, transcoding MPEG 2 into MPEG 4, and converting a stored MPEG Packetized Elementary Stream (PES) into an MPEG Program Stream. Each distinct conversion or decryption would be handled by a *Format Conversion* task. *Format Conversion* threads of that task would be multiple instances of the same task preparing distinct streams for various clients expecting the same data structure. The ultimate goal of each *Format Conversion* task or thread is to put media stream packets on a transmission queue handled by the *Server* thread in communication with the destination client. In effect, a *Format Conversion* module makes the correct data available and a *Server* module sends it using the protocol specified by the client. Of course, the more complex the *Format Conversions* to be done and the more *Format Conversions* to be simultaneously supported, the greater the need for power and resources in the settop box. The good news is that modern economical processors are powerful enough to feed a couple of clients with nominally converted data streams.

A *Format Conversion* task may also be an appropriate place to perform Dynamic Program Assembly (DPA). Dynamic Program Assembly is the process of building a program in real-time from multiple live and/or stored segments. An example of this is the local insertion of highly focused and customized advertisements into a real-time broadcast. Here, generic (place-holder) advertisements in the broadcast are mapped out and replaced by advertisements determined to be of special interest to the owner of the settop. The replacement advertisement could have been received earlier and previously stored on the hard drive or they could be arriving live on an alternate multiplexed transmission stream. Regardless, this dynamically assembled (and perhaps totally unique) media stream is then sent to the associated unsuspecting and indifferent client.

Server

The *Server* is the system component responsible for delivering a media stream to a client. Each active client is associated with a specific *Server* instance. There are three basic components of each *Server* instance:

<u>The Transmission Queue</u>
The Transmission Queue is fed by a *Format Conversion* task or thread and is handled by a *Server* thread. This queue provides the media stream input to the *Server*. It is expected that the queue provides correctly formatted data in the correct sequence. The *Server*'s job is to send this data using the *Client* specified protocol.

<u>The Protocol Engine</u>
It is the Protocol Engine that maintains the connection with the *Client*. The connection may be point-to-point or multicast and it may be “push” or “pull”. In a “pull” connection the *Client* requests each new block of data. An example of this could be some type media player (the *Client*) connected by HTTP to the *Server*. The media stream itself tells the *Client* how fast to consume the data. As the media data is consumed, buffer space is freed and the *Client* requests additional data. In a “push” connection, the *Server* throttles the data stream and sends each new block of data at the appropriate time and rate. The *Client* does relatively little buffering and consumes the data at essentially the same rate it arrives. An example of this would be a media player that joined a multicast transmission. The *Server* determines the rate at which the multicast is sent. All listening *Clients* must do their best to keep up.

Examples of a Protocol Engine include HTTP, RTP, and of course a large number of proprietary implementations.

<u>The Communications Pipe</u>
The Communications Pipe connects the *Server* to the client. This connection can be virtual via standard telecommunications protocols such as TCP or UDP, or could be a physical connection across a USB cable, hardware bus, or with a custom ASIC. The important point is that this is only one more link in the chain that connects data stored on the hard disk to the *Client*.

Client

There may be multiple *Client* instances as well as many diverse types of clients. A *Client* may be “embedded” running in the same box as the server or in a settop box in another room. A client may also be running on a PC, MAC, PDA, etc. in another room somewhere nearby. For the purposes of our Embedded Streaming Media Server, *Clients* running remotely (i.e. through the Internet) have reduced capabilities and will not be able to receive all forms of media available to the local *Client*. Essentially, a

Client may be any device or interface capable of displaying the stored media. The basic components of a client include:

A User Interface
The appearance and capabilities of the user interface are highly dependent on the platform on which the *Client* runs. Examples may be simple VCR style icons or modest menus. At the other extreme may be sophisticated interfaces with changeable "skins". The important issue is that the system as a whole and specifically the connected *Format Conversion* module properly interact with the commands issued by the user interface. It is worth noting that PlayLists are a form of user interface. A PlayList may do much more than specify a sequence of programs to play. A properly designed PlayList implementation allows looping on a movie or playing segments of a movie out of order. A PlayList may even be used to dynamically insert into the media stream advertisements specifically targeting the owner of the *Client*.

A CODEC
The CODEC consumes the multimedia data and presents audio and video to the user. This paper is not particularly concerned with details of the CODEC. It is sufficient that it is compatible with the output of the *Format Conversion* module connected to this *Client*. Compatibility requires the basic need to understand the media encoding and the ability to achieve audio/video synchronization under all characteristics and variations of the media stream. It would also be desirable that the CODEC support special features needed by various play modes (pause, fast forward, rewind, etc.). In the case of *Server* pushed data, the CODEC must be able to keep up with the arriving stream.

A Display Device
The Display Device may be a large screen high definition TV, the tiny screen of a cell phone, or anything in-between. In all cases, the CODEC, the User Interface, and the *Format Conversion* module associated with this *Client* must all agree on the capabilities of the display.

System Issues

A typical application of the Embedded Streaming Media Server described in this paper might be recording a movie, simultaneously presenting it real-time to an attached TV, and also presenting a previously recorded movie remotely on another TV. One would expect that viewers of both televisions be able to independently pause or rewind their presentation. At the most fundamental level, the ESMS is unavoidably a multi-output device. Moreover, each output must be distinct and autonomous. It could literally be true that the same movie is being watched in the family room and bedroom. However, the instance being viewed in the bedroom may lag that of the family room by one second - or by one hour. There is no need to "aggregate" viewers into watching a single stream. There is no need for one viewer to see a frozen image when the other pauses the program. A well designed relatively economical platform is certainly capable of streaming two or three or more independent programs throughout a home or apartment complex.

Task Priorities and Resource Allocation

A good ESMS system design allocates sufficient buffering, processor horsepower, disk bandwidth, etc. so no stream ever starves of data or processing. The challenge of an embedded system designer is to minimize the system cost while maintaining adequate data processing capabilities. There are no special challenges associated with establishing the task priorities for an Embedded Streaming Media Server. It is just an embedded system. It is reasonable that *Acquisition* modules responsible for capturing real-time data run at a high priority. It is also reasonable that a user interface run at a relatively low priority. *Format Conversion* and *Client* modules should receive a medium priority. Early in the design process a decision should be made with respect to guaranteeing the availability of sufficient system resources to assure that a new *Client* can be fully serviced. Guaranteeing that sufficient worst case resources are available can be pessimistic for a consumer priced device. Denying a new *Client* that could be well served under typical circumstances may be unacceptably pessimistic and costly for a consumer product. A more appropriate consumer product model may be that of graceful degradation where responsiveness slows when system resources get tight. Setting the criteria of acceptable degradation is a difficult responsibility since it may present a very strong image of quality. Some careful system analysis and consideration of your corporate image is involved in setting this quality level.

State Management

There are quite a large number of operational states in an Embedded Streaming Media Server. Tracking the progress of each streaming instance, for example, involves knowing the next disk block, the last disk block in the file (so we know when we are done), whether we are playing or paused, the ID of the stream destination, etc. Using global variables for this tracking seems inappropriate since the information may be different for each active instance. It is best if an instance of state management variables is allocated to each active module. This implies that each task or thread of a task must allocated and initialize a good sized block of tracking variables. In addition, methods of accessing this information must be created to support the need for some of this information by other system components. The good news is that all this information is being created and accessed at video speeds. Ah!! You wanted challenges like this when you decided to become an engineer.

Component Qualification

Occasionally a great deal of effort is exerted to qualify a few components determined to be possible bottlenecks. The performance of the hard disk, for example, is certainly critical to the overall operation of the product. It seems very reasonable to develop a comprehensive disk drive stress test to insure it has adequate performance. However, some moderation and restraint in this area may be beneficial. All components must be reliable and tests must be run to validate that they are. It may be questionable whether accurate performance stress tests can be developed for some of the components. Certainly tests can be developed that show differences between different model hard drives. It may be difficult to prove that these differences are relevant to the highly statistically variant environment of an Embedded Streaming Media Server. It may be more difficult to justify paying significantly more for this dubious performance gain. To use an analogy, the ESMS is like a football team. The hard drive is like the quarterback. Having a great quarterback is wonderful, but it is more important to have a great team. The best disk drive can be ruined if surrounded by a bad design. A great defense and a good running game can get a lot of mileage out of an economical quarterback. Make sure the system design is sound. A great design should be able to work with any (reliable) hard drive.

Error Conditions

This is not your grandmother's television. A staggering number of error conditions are possible. The disk may be full when an *Acquisition* module tries to run. The user in the bedroom may try to delete a program being watched by someone in the family room. The PC receiving the stream of music was just turned off. The owner just unplugged the box before leaving for the beach. Billy accidentally deleted a half hour of his favorite movie while editing commercials. Jane wants to watch TV but forgot her parental control password. Etcetera, Etcetera, Etcetera…

An Embedded Streaming Media Server is a very complex device being operated by untrained personnel in the most hostile environment imaginable - the home. Children, adults, the family pet, and even your brother-in-law will tamper with this device. Early design consideration must be given to detecting and gracefully respond to all error conditions. Where will error logs be stored and how will they be retrieved? What actions are legal under what conditions and what must be prevented at all costs? Will it be possible to fix bugs with software downloads? Will a maintenance staff be kept on the project to produce needed downloads in the future? Who will look at retrieved error logs?

Conclusions

In the near future settop box manufactures will provide consumer priced digital recording boxes capable of receiving, storing, and serving media streams to multiple household clients. Skilled embedded engineers have already designed products that receive hundreds of channels and simultaneously record several of them. Soon, economical consumer products will be capable of much more. The recorded content will be streamed to multiple and varied clients around the house. It is likely that an entire industry of third party providers will evolve to help consumers aggregate, edit, and customize their media treasures. A few terabytes of storage will accommodate thousands of movies and songs, instructional and educational videos, faded childhood pictures and camcorder memories. All of this will be available on TV sets, stereos, laptops, PDAs, and maybe even cell phones. It is the skill and vision of embedded engineers that will enable these revolutionary consumer products. The pioneers will define standards, features, and operational paradigms for an entire generation of products. Soon, boredom will be a thing of the past. Something interesting and entertaining will be instantly available from any number of devices scattered around your house.