Ollscoil na Gaillimhe
University of Galway

Provided by the author(s) and University of Galway in accordance with publisher policies. Please cite the published version when available.

Title	Architecture of a PVR appliance with 'Long Tail' internet-TV capabilities
Author(s)	Corcoran, Peter; Callaly, Frank
Publication Date	2006
Publication Information	F. Callaly, P. Corcoran, (2006) " Architecture of a PVR appliance with 'Long Tail' internet-TV capabilities", IEEE Transactions on Consumer Electronics, Vol. 52, No. 2, Page(s) 454-459.
Publisher	IEEE
Item record	http://hdl.handle.net/10379/291

Downloaded 2023-03-27T19:51:13Z

Some rights reserved. For more information, please see the item record link above.

Architecture of a PVR Appliance with 'Long-Tail' Internet-TV Capabilities

F. Callaly and P. Corcoran, *Member, IEEE*

Abstract — *The design and implementation of a networked PVR appliance incorporating support for Internet-TV is described. The appliance incorporates support for the bittorrent protocol and employs user tools for content location, management and scheduling which encourage background downloading of content and it is particularly suited for "long-tail" content distribution applications. The standard bittorrent algorithm has been modified to allow client bandwidth and content storage to be managed from a central content server. With this modified protocol each PVR can function as a combined broadcast and storage node in an Internet-wide distribution system*[1].

Index Terms — PVR, content distribution, Internet TV.

I. Introduction

Personal video recorders have revolutionized the way consumers watch television. Time shifting allows programs to be watched when it suits the viewer rather than the broadcaster. This new added-value functionality is changing the way consumers treat broadcast media sources. The providers of such sources no longer have the clear ability to differentiate their services based on the real-time nature of broadcasting.

Interestingly, a new generation of digital services provider sees opportunities in providing similar real-time services over digital distribution networks [1]. There is not doubt that the increased bandwidth of networks will make make such services possible but we question if the availability of real-time content will remain as compelling as it has done over the last 50 years, thus enabling the providers of real-time content to charge a premium for such services.

An alternative perspective is that of the "long-tail" of content distribution. This implies supplying specialized niche content to a far smaller audience than conventional broadcasting. By employing the Internet as a distribution mechanism such services become economically viable. Many examples of success in the Internet marketplace are based on such long-tail models: Google makes most of its revenues from small advertisers and eBay from large volumes of niche or one-off products/sales. Is this also where the future lies for the broadcast industry?

In this paper we propose a modified PVR with support for network torrents. From a user perspective this allows content to be harvested from a variety of network services. From the perspective of the service provider we propose a modification of the *bittorrent* architecture to allow content providers to broker distributed storage space on clients that have uploaded their content. This allows each PVR to function as a store & broadcast node for service providers. In effect a network of PVRs that conforms to this architecture can act as a cost-effective means of content distribution, yet retaining certain useful aspects of more conventional broadcast services.

II. PVR Overview

In this paper we describe the software architecture of a PVR appliance which is specifically designed to support "long-tail" content distribution. This appliance provides conventional PVR-style user interfaces but with additional features which enhance its functionality and assist a viewer in locating, registering and managing content sources.

A. Peer-to-Peer Networking Support

The appliance supports *bittorrent* peer-to-peer (P2P) networking [2] and can manage and control access to similar networking technologies using a standardized plugin interface. Users can access P2P content using a specialized browser for RSS feeds. Once content is located its status can be monitored and, subject to certain limitations, the user can adjust the time for which content will be stored and made available to other users.

B. UPnP Support

The appliance is UPnP compatible, allowing it to be easily integrated with a WinXP desktop PC, or with a home network which supports UPnP devices. Because it supports more sophisticated content access mechanisms than conventional UPnP appliances this requires some modifications to the standard UPnP class hierarchy. Several examples of required modifications are given in section III below.

C. Flexible Content Access Mechanisms

Content can be loaded in real-time or background modes with different levels of QoS. Typically, real-time content will be obtained from the local home network via, for example, a cable-TV set-top box. Content from the Internet, in particular content torrents, will normally be downloaded in a background mode, although we have initiated some investigations into micro-torrents which allow large content files to be downloaded in smaller sequential blocks. This mechanism trades off some of the benefits of using *bittorrent* but allows earlier viewing of larger content files.

In our prototype it is assumed that this content will be streamed in MPEG2 or MPEG4 format. The preferred streaming mode is a unicast TCP/IP stream, although UDP and RTP are also supported in unicast mode and multicast addresses may be configured to allow local rebroadcasting of content.

[1] Peter Corcoran is Director of Research at the Consumer Electronics Research Group of NUI, Galway (e-mail: peter.corcoran@nuigalway.ie).

Frank Callaly is a PhD student at the Consumer Electronics Research Group of NUI, Galway (e-mail: frankc@wuzwuz.nuigalway.ie).

Manuscript received January 25, 2006 0098 3063/06/$20.00 © 2006 IEEE

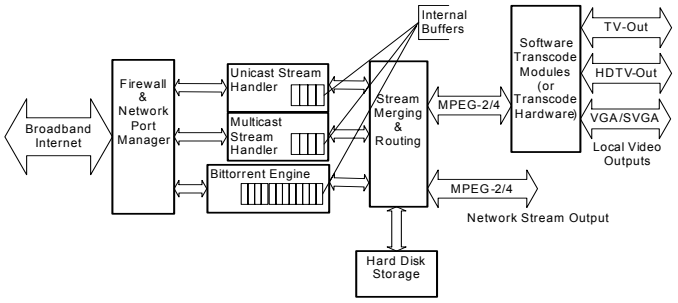


Fig 1: Internal software architecture of PVR appliance; modules for searching and managing content not shown

D. Transcode & LAN Rebroadcast Tools

The appliance is network enabled and incorporates transcoding and rebroadcast capabilities; this allows it to be used for "harvesting" content from a broadband connection which can be redistributed to other networked appliances over a home network. Content will typically be transcoded to MPEG4 format for rebroadcast over a home WLAN, although other video encoding standards are supported.

An overview of the architecture and software components of the device is given in ***Fig 1***. The appliance in implemented using an embedded Linux operating system with integrated UPnP system components. The principle system software components include unicast and multicast stream handlers and a *bittorrent* engine for receiving content and a range of transcoder and buffering components which allow it to function as a multi-functional content storage and management appliance.

E. Appliance Hardware Platform & OS

Our appliance is based on the latest mini-ITX motherboards which include MPEG4 hardware decoding and support a Linux 2.6 kernel. In addition a TV decoding card with MPEG2 encoding hardware is incorporated in these appliances which also have DVD drives and, typically, an 80GB hard disk for content storage.

Primary network connectivity is via an 802.11g WLAN card and there is support for both wireless and IR remote controls. Standard TV sets can be driven or a SVGA output is used for connection to Plasma screens.

F. Software Development Environment

A number of openly available libraries were used in device development. Intel's UPnP library [6] was used to as the basis for the middleware components of the devices. A number of openly available media codecs were also used to encode/decode content, in particular the libavcodec and libavformat libraries from the ffmpeg [7] project were used, Developing new codecs for each of the supported media formats would be quite impractical when prototyping, as it would require a large development effort. The video display engine was based on the freely available Xine [8] video engine.

III. PVR APPLIANCE MODEL

A. Generic Appliance Models

Earlier research work [3] has demonstrated that practically all networked A/V appliance configurations can be designed and prototyped as combinations of three generic types of networked A/V appliance namely:

1) A/V streaming servers:

Also known as MediaServers in UPnP parlance; these are devices which provide access to media content. The content may be derived from a number of sources including live terrestrial or satellite TV, removable or fixed storage devices (e.g. DVD, HDD) or other networked A/V servers (e.g. Internet servers). These devices allow the user to browse the available content so that items of interest can be easily located. They also provide an interface for streaming clients to access the content.

2) A/V streaming clients:

These are generally display devices, these are known as MediaRenderers in UPnP and are charged with the task of converting and/or filtering the received A/V stream to optimize presentation on a display console.

3) A/V transcoding appliances:

These devices convert a data stream from one format to another; for example it might be necessary to convert a high resolution MPEG2 stream to a low resolution MPEG4 stream for display on a handheld device. This type of device may also be used as a buffer, so that a user can be pause or rewind a live TV stream, even if this functionality is not supported by the originating streaming server.

B. PVR System Components

These are illustrated in ***Fig 2*** below. A base PVR can be built from a standardized *transcoder* module and a *media server* module. Content can be accessed from a range of different sources as illustrated. These all appear as common content under UPnP and a content-specific plugin is responsible for handling the differences between each type of content.

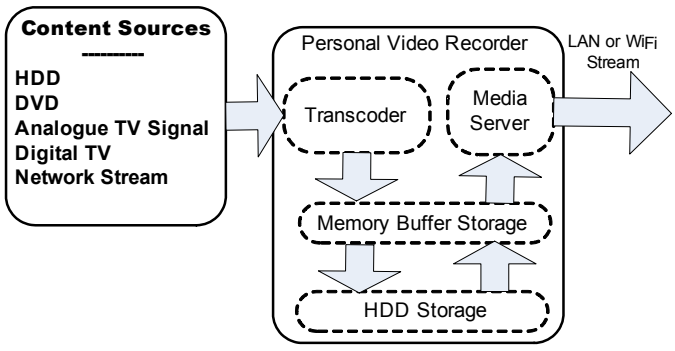


Fig 2:PVR System Components

It is useful, although not essential, to have a RAM buffer between the hard disk storage and these higher level software components. This facilitates various trick play modes such as rewind, slow motion replay and skipping blocks of advertising. Higher level components of the RAM buffer may be incorporated into the transcoder or media server as discussed in the next section.

C. Transcoder Components

These are illustrated in ***Fig 3*** below. The main transcoding engine is implemented in Python which allows rapid prototyping of the transcoder workflow. Lower level components such as the A/V codecs and the A/V multiplexer and demultiplexer modules are built using open-source library components with Python or C/C++ wrappers as appropriate. The A/V buffer interface provides access to the RAM buffer extensions of the filesystem described in the previous section. This is implemented independently from the transcoder module.

A number of reusable software components were developed specifically for the generic transcoder module. An RTSP [9] video on demand (VOD) server was developed which is used by server devices to export content to clients. Using RTSP, a TCP connection is initially established between the client and the server, this connection is used to setup and control the stream. The A/V stream is then sent over a separate UDP connection, As the UDP data is being sent over a wireless network, the server will attempt to use the most robust format that the client can accept. This may be RTP packets or MPEG transport stream packets.

A specialized HTTP server was also developed, this server can be used in a similar manner as the RTSP server. It is used by clients that do not support the RTSP protocol. Both of these servers read A/V streams from loadable access modules. Each access module allows streams to be read from a particular source, e.g. hard drive, video capture card, CD, DVD. This means that they can be extended to read from new sources as they arise.

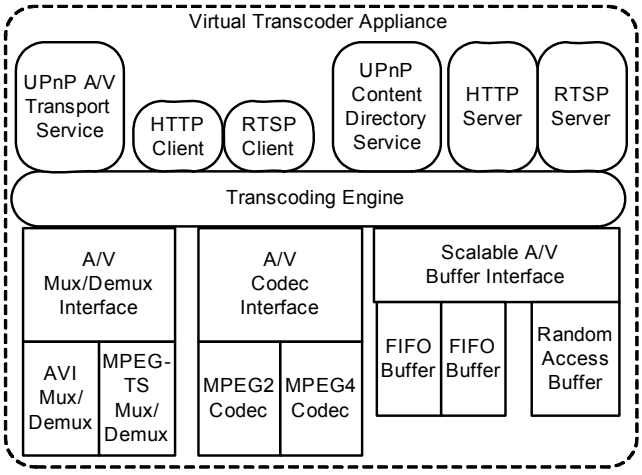
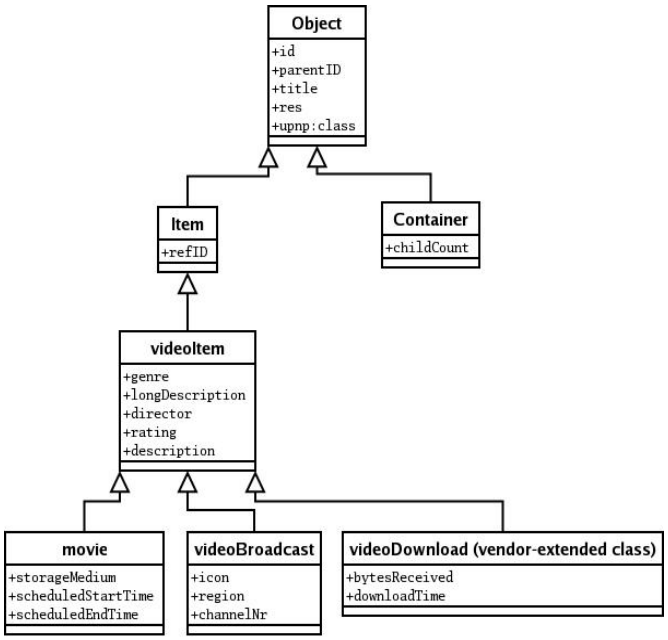


Fig 3: Transcoder Subsystem Software Components

IV. MODIFICATIONS TO UPNP

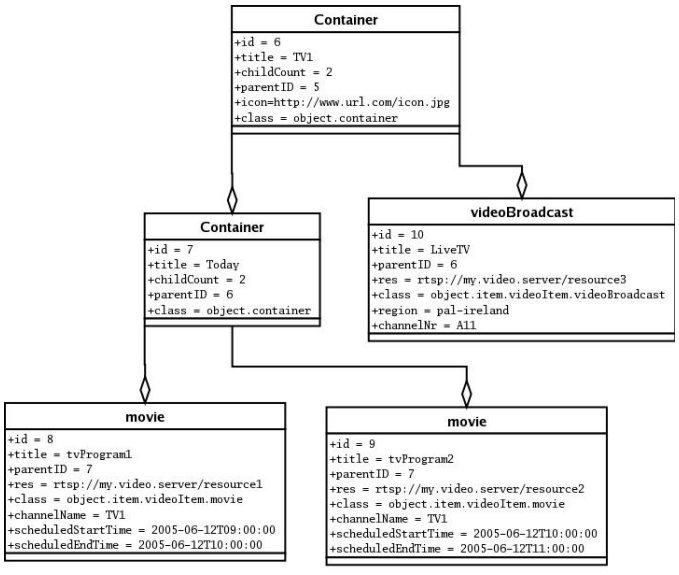
The standard UPnP class structures need to be modified for our appliance. Some examples are illustrated in ***Fig 4*** below. In particular ***Fig 4(a)*** illustrates how the content directory class hierarchy needs to be modified to take account of background downloading of video content and the additional capability for local rebroadcast of a video stream using multicasting.

Additional modifications to the content tree section are illustrated in ***Fig 4(b)*** where the standard content class for movies is extended to incorporate a videoBroadcast class object.



UPnP Content Directory Class Hierarchy (extended)

Fig 4(a): Extended UPnP content directory class hierarchy.



Sample Content Tree for live tv or live internet stream

Fig 4(b): Content tree section for downloadable content.

V. NETWORK OPERATION

The operation of the appliance on a typical network is illustrated in ***Fig 5*** below. The main system component exports software interfaces to content server and renderer modules. These may run on the same physical device as the

main PVR appliance, but they may also interface with separate media server and renderer appliances. The media server appliance may provide real-time content by, for example, decoding a conventional TV signal using a tuner card and re-encoding it as an MPEG digital stream.

The simplest form of content renderer is a laptop computer running a video player which can process real-time streamed content. An alternative content renderer can be provided using a TV set and a media adapter. This allows MPEG content to be streamed to a conventional analog TV set. PVR functionality is provided by buffering media streams on a local hard disk drive.

Our appliance supports several of the more popular media adapters available on the market.

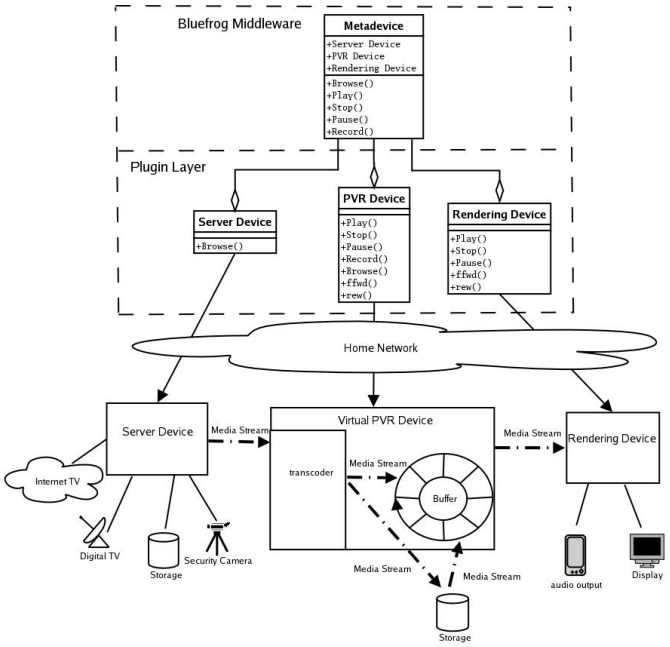


Fig 5: Overview of the PVR Appliance.

The bittorrent protocol is a peer-to-peer TCP based protocol which has been designed for distributed file sharing and distribution over the Internet. *Bittorrent* breaks files down into smaller fragments, typically a quarter of a megabyte (256 KB) in size. Peers download missing fragments from each other and upload those that they already have to peers that request them. The protocol is 'smart' enough to choose the peer with the best network connections for the fragments that it's requesting.

A. The Bittorrent Architecture

To increase the overall efficiency of a "content swarm" (the ad-hoc P2P network temporarily created to distribute a particular file), the bittorrent clients request from their peers the fragments that are most rare; in other words, the fragments that are available on the least number of peers, making most fragments available widely across many machines and avoiding bottlenecks. The file fragments are not usually downloaded in sequential order and need to be reassembled by the receiving machine.

It is important to note that clients start uploading fragments to their peers before the entire file is downloaded. Sharing by each peer therefore begins when the first complete segment is downloaded and can begin to be uploaded if another peer requests it. This scheme is particularly useful for trading large files such as videos and operating systems. This is contrasted with conventional file serving where high demand can lead to saturation of the host's resources as the consumption of bandwidth to transfer the file to many requesting downloaders surges.

With *bittorrent*, high demand can actually increase throughput as more bandwidth and additional "seeds" of the file become available to the group.

B. Bittorrent Terminology

Torrent - a torrent can mean either a .torrent metadata file or all files described by it, depending on context. The torrent file contains metadata about all the files it makes downloadable, including their names and sizes and checksums of all pieces in the torrent. It also contains the address of a tracker that coordinates communication between the peers in the swarm.

Swarm - together, all users sharing a torrent are called a swarm. Six peers and two seeds make a swarm of eight.

Peer - a peer is one instance of a *bittorrent* client running on a computer on the Internet that you connect to and transfer data. Usually a peer does not have the complete file, but only parts of it, however, 'peer' can be used to refer to any participant in the swarm (in this case, also known as a 'client').

Seed - a seed is a peer that has a complete copy of the torrent and still offers it for upload. The more seeds there are, the better the chances are for completion of the file.

Leech - a leech is usually a peer who has a negative effect on the swarm by having a very poor share ratio - in other words, downloading much more than they upload. Most leeches are users on asynchronous Internet connections who do not leave their *bittorrent* client open to seed the file after their download has completed. However, some leeches intentionally hurt the swarm to avoid uploading by using modified clients or excessively limiting their upload speed.

Tracker - a tracker is a server that keeps track of which seeds and peers are in the swarm. Clients report information to the tracker periodically and in exchange receive information about other clients that they can connect to. The tracker is not directly involved in the data transfer and does not have a copy of the file.

C. Files Sharing with Bittorrent

To share a file using the *bittorrent* protocol, a user creates a .torrent file, a small "pointer" file that contains: (i) the filename, size, and the hash of each block in the file (which allows users to make sure they are downloading the real thing); (ii) the address of a "tracker" server and (iii) other data such as client instructions.

The torrent file is then distributed to users, often via email or placed on a website. The *bittorrent* client is started as a "seed node", allowing other users to connect and begin

downloading. When other users finish downloading the entire file, they can optionally "reseed" it--becoming an additional source for the file. One outcome of this approach is that if all seeds are taken offline, the file may no longer be available for download, even if a client has a copy of the torrent file. However, everyone can eventually get the complete file as long as there is at least one distributed copy of the file, even if there are no seeds.

Downloading with the *bittorrent* protocol is straightforward. Each person who wants to download the file first downloads the torrent and opens it in the *bittorrent* client software. The torrent file tells the client the address of the tracker, which, in turn, maintains a log of which users are downloading the file and where the file and its fragments reside. For each available source, the client considers which blocks of the file are available and then requests the rarest block it does not yet have. This makes it more likely that peers will have blocks to exchange. As soon as the client finishes importing a block, it hashes it to make sure that the block matches what the torrent file said it should be. Then it begins looking for someone to upload the block to.

D. Bittorrent Modifications for Long-Tail Broadcasting

Although the *bittorrent* protocol is a very well designed and engineered protocol it suffers from some disadvantages in practical usage scenarios. It does not provide, for example, any mechanism for checking the integrity[2] of the data files which are downloaded from the Internet. All that *bittorrent* guarantees is that you will get the original file referred to by the torrent metadata. But it is possible that this original file was corrupted or even behaves maliciously when activated.

As the *bittorrent* protocol is widely used to distribute illegal copies of music and video files certain content providers have fought back by creating bogus client software and bogus torrents which attempt to disrupt or sabotage content swarms. In short there is no means to indicate the quality of *bittorrent* content prior to its download. However this difficulty can be readily overcome by incorporating a means of encoding and/or digitally signing the distributed content at source [5].

A second disadvantage is that clients may connect and disconnect at will from a content swarm and may set their upload and download rates at fixed values. This means that clients may download files from the swarm at a faster rate than they upload (or rebroadcast) them. Clients that "take" more than they "give" from the swarm are known as "leeches". Also, as the number of clients in a swarm decreases it takes progressively longer for the remaining clients to "get all the pieces". This tends to happen naturally as the popularity of some distributed content begins to wane.

A third disadvantage is that a *bittorrent* client/peer downloads data in no particular order. Thus it is not possible to begin playing a video when it is 50% downloaded – you must wait until the complete file has been obtained. From the perspective of service provider this is a significant drawback. It means that *bittorrent* can only supply data to a PVR in a "record" mode rather than in a "live" mode. This presents one of the challenges that this research has uncovered – how can we modify the *bittorrent* architecture and protocol to facilitate a more timely and orderly delivery of content? We will not answer this question here, but rather leave it as a subject for future research.

In ***Fig 6*** we illustrate how a service provider can control and feed a content swarm of PVRs which support our modified *bittorrent* protocol. Note that in an unfettered Internet environment the computer which hosts the *tracker* would not normally seed content as well. In fact this is one of the advantages of *bittorrent*, particularly for illegal file sharing applications.

In our system we host both *tracker* and *seeding engine* on the same server. We have also removed most of the client-side flexibility provided by the *bittorrent* protocol in favor of adding client control functionality to the *tracker*. The idea here is that clients cannot choose to throttle back on their upload bandwidth which is, instead, controlled and managed from a centralized server. This allows the server to prevent client PVRs from leeching a content swarm.

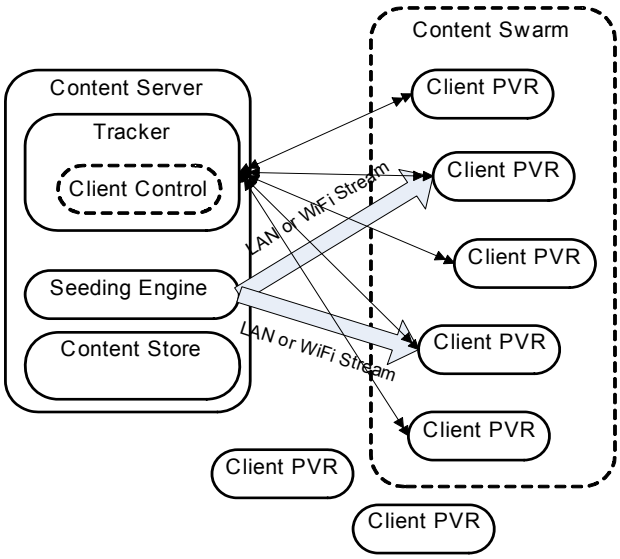


Fig 6: Content Server Feeding and Controlling a Content Swarm

We still allow a limited level of control at the client side, for example a client can choose to reduce its upload bandwidth but then it must hold data for a longer period of time after the full torrent is downloaded. This is implemented by means of a delete queue where a torrent may still be held on the hard drive after the user has chosen to delete it until a certain minimal time has elapsed.

[2] Bittorrent does perform integrity checking on the raw data blocks of a file it is handling, but what is meant here is that there is no way to confirm that an audio/video file is actually MPEG compliant until it is completely downloaded; bittorrent does not incorporate a means of encrypting, encoding, or digitally signing data to guarantee that content has not been tampered with.

VII. Modified PVR Architecture

The above considerations have led to the PVR architecture illustrated in ***Fig 7*** below. This is modified over the generic PVR illustrated in ***Fig 2*** above.

A. Torrent Manager

This component is distinct from the main transcoder module. It is implemented with a similar user-interface to the standard UPnP content browser, but at present operates in a permanent record mode. After a torrent is fully uploaded it is then transferred to the conventional HDD content list and appears, to the user, as a standard MPEG file. This implementation is not entirely satisfactory as some torrents may run out of supporting peers and may never complete.

Our goal in providing centralized server control over clients was to try and overcome this disadvantage of a conventional content swarm. Thus, when a torrent remains incomplete beyond a certain time limit it informs the central server which will re-seed the missing data. In a more sophisticated implementation it should be practical to query other PVR clients to determine if the missing content is available outside of the main server.

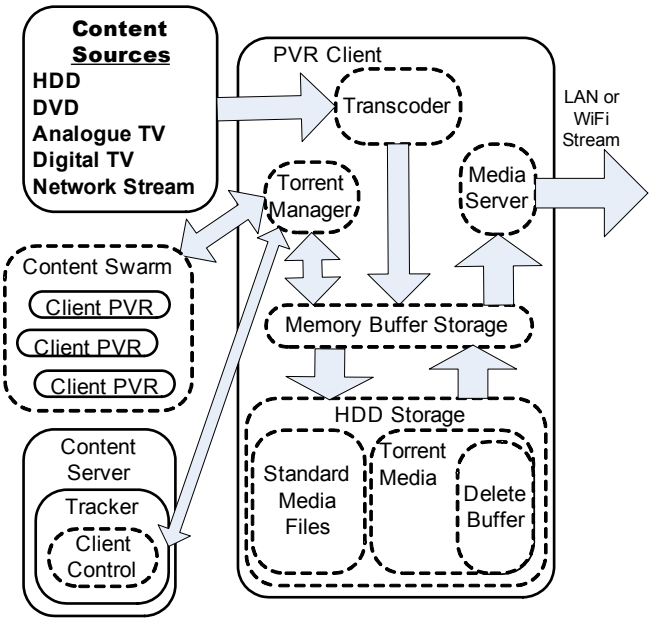


Fig 7: PVR Architecture incorporating Bittorrent Modules.

B. Filesystem Modifications

A second aspect of the modifier PVR is that the file-system is modified to separate torrent and non-torrent data files. This is partly to allow a more detailed auditing of the torrent data storage and partly to allow for a direct association between the metadata for a torrent and the actual data file itself. The latter aspect allows files to be kept on the hard disk after a user has deleted them and to use these files for re-seeding a content swarm. Information about such "retained" files is sent from the client to the central content server.

VIII. Conclusions

Changes in TV viewing habits introduced by PVRs mean that many people no longer view content at a fixed time. In fact the viewing experience for many users of PVRs which are linked with a content service is now a two stage process: (i) select and reserve content and (ii) view the recorded content.

This has led us to investigate the addition of a *bittorrent* based content service to a conventional PVR appliance. As *bittorrent* is a distributed service we found that it was important to consider centralizing certain control aspects of the protocol on a central server. This provides additional levels of control over a content swarm which are important to service providers. In particular it allow better control of QoS aspects and allows the service provider to exclude or eliminate leeching peers from the swarm.

One difficulty we experienced was in performing meaningful testing of modifications to the torrent related aspects of the system operation. Because alterations were made to the clients we were limited to testing content swarms with 4-6 client appliances and these were mostly located on the same WAN segment. Clearly this does not provide an adequate simulation of a broader Internet environment and this issue needs to be considered in further work. Nevertheless our conclusion is that a *bittorrent* component can provide useful PVR services for consumer appliances. It allows a new low-cost content distribution mechanism to be easily added to any Internet connected PVR appliance.

Acknowledgment

This research was funded under the *Technology Development Phase* of the Enterprise-Ireland Commercialization Fund.

References

[1] Microsoft TV IPTV Edition, http://www.microsoft.com/tv/content/Solutions/IPTV/mstv_IPTV_Overview.mspx
[2] Bram Cohen, "Incentives Build Robustness in Bittorrent", in Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, June 5--6, 2003.
[3] P. Corcoran and F. Callaly, "Rapid Prototyping of Networked A/V CE Appliances", IEEE Eurocon 2005, Belgrade, Nov 2005.
[4] P. Corcoran, A. Cucos and F. Callaly, "Home Networking Middleware Infrastructure for Improved Audio/Video Appliance Functionality and Interoperability", IEEE Eurocon 2005, Belgrade, Nov 2005.
[5] P. Corcoran and A. Cucos, "Techniques for Securing Multimedia Content in Consumer Electronic Appliances using Biometric Signatures", *IEEE Trans. Consumer Electronics*, Volume 51, Issue 2, May 2005 p545 – 551.
[6] Linux SDK for UPnP Devices, http://upnp.sourceforge.net/
[7] ffmpeg multimedia system, http://ffmpeg.sourceforge.net/
[8] Xine video player, http://xine.sourceforge.net/
[9] *"Real Time Streaming Protocol"*, Internet Engineering Task Force, RFC 2326

Peter Corcoran received the BAI (Electronic Engineering) and BA (Math's) degrees from Trinity College Dublin in 1984. He continued his studies at TCD and was awarded a Ph.D. in Electronic Engineering for research work in the field of Dielectric Liquids. In 1986 he was appointed to a lectureship in Electronic Engineering at NUI, Galway.

Frank Callaly received the B.Eng. (Electronic & Computer Engineering) degree from the National University of Ireland, Galway in 2003. He continued his studies with the Consumer Electronics Research Group at the National University of Ireland, Galway and is currently pursuing a Ph.D. degree in Electronic Engineering. His current research interests include digital video encoding, multimedia delivery techniques and home networking.