DISCOVR: Distributed Collaborative Video Recorder

Jin Li and Cheng Huang
Communication and Collaboration Systems, Microsoft Research
One Microsoft Way, Bld. 113, Redmond, WA 98052
Email: {jinl,chengh}@microsoft.com

Abstract

The paper describes DISCOVR, a distributed collaborative video recorder. DISCOVR is a P2P application that combines asynchronous file sharing with synchronous on-demand media streaming. DISCOVR uses a flat entity ID space, with the entity being any of the media file, header, mega packets, index and metadata. All DISCOVR entities may be asynchronously or synchronously distributed. DISCOVR adopts a sender-driven priority based sharing protocol. If the user is on-demand viewing a media file, those packets that are to be viewed in the near future will be put on the synchronous access list, which prompts its connected peers and the peers that are indirectly connected to fulfill the distribution of the on-demand packets in high priority. By letting the peers engage in both asynchronous sharing and synchronous on-demand streaming, DISCOVR promotes the peers to remain online longer, thus improve the availability of the P2P system and the overall performance.

1. Introduction

The popularity of peer-to-peer (P2P) application is on the rise. Compared with the client-server architecture (such as the web server and the media server), there is no absolute notion of clients or servers in a P2P network. Instead, the peer nodes function as both clients and servers to the other nodes on the network. In a P2P network, as the demand of the service grows, the capacity (network bandwidth, storage capacity and computing power) of the network grows as well. Moreover, the capacity of the P2P network is provided collectively by all the peers in the network. This is in sharp contrast to a client-server architecture, where the capacity is fixed and paid for by the server.

The first major P2P application for the masses was the Napster P2P file sharing. Although the original Napster service has been shut down by court order for copyright violations, its popularity opens the floodgate of the decentralized P2P applications. Some popular P2P file sharing applications are Gnutella[1], eDonkey, FastTrack and BitTorrent. Gnutella was the earliest development, with completely decentralized network and broadcast search message. The design goal of Gnutella is that it should be virtually impossible to shut down. However, the growing surge in popularity reveals the limits of the Gnutella protocol's scalability, as search message overwhelms data exchanged and wasted huge resource in the network. FastTrack[2][3] is a P2P protocol used by the Kazaa and its variants (Grokster and iMesh). FastTrack shares files in P2P fashion, but uses supernodes for semi-centralized search and indexing. eDonkey[4], another popular file sharing networks, utilizes servers as communication hubs. A recent major development in P2P file sharing protocol is BitTorrent[5]. Written by Bram Cohen, BitTorrent breaks the file into smaller fragments, and distributes the fragments in a P2P fashion. To use BitTorrent, each user first downloads a torrent file, which contained the address of a tracker node and the hashes of the fragments. The hash prevents malicious attacker from corrupting the P2P content in distribution. The tracker node maintains a log of which users are downloading the file and what their progresses are in the download[1]. BitTorrent uses two unique mechanisms for efficient P2P sharing. First, it distributes the fragments that are the "local rarest" to the peers. This ensures that the fragments distributed to the peers can be redistributed to the other peers, thus fully uses the peer's bandwidth resource. Second, a BitTorrent[5] peer uploads to the connecting peers that are also uploading content to itself. This Tit-for-Tat strategy gets rid of the leech behavior that plaques the common P2P networks.

Napster, Gnutella, eDonkey, FastTrack and BitTorrent distribute content in asynchronous mode. The file is shared piece by piece. And all file pieces have to be delivered before the entire file can be viewed. This leads to long waiting time if the file to be distributed is large, e.g., a digital music or movie. Recently, a number of works have investigated synchronous P2P content distribution, namely broadcasting / streaming of the media files. The basic idea is to construct an application-level multicast (ALM) tree that covers all receiving peers. The media server then broadcasts the media packets through the ALM tree. For example, PeerCast [6] constructs a distribution tree rooted at the sender for a live media streaming session. A new receiver joins by traversing the tree starting at the root until it reaches a node with sufficient remaining capacity. NICE [7] and Zigzag [8] use hierarchical distribution trees and therefore can accommodate a large number of peers. To improve reliability and ensure that the leaf nodes do not become leeches, multiple ALM trees or a distribution mesh can be constructed. Each peer then receives different portions (stripes) of the stream from multiple ALM trees. CoopNet [10] employs multi-description coding and constructs multiple distribution trees (one tree for each description) spanning all participants. SplitStream [11] provides a cooperative infrastructure that can be used to distribute large files (e.g., software updates) as well as streaming media. CoolStreaming [9] uses a BitTorrent like protocol that forces data exchange during the media broadcast. These are just a few examples of the many P2P media broadcast systems.

In this work, we develop DISCOVR, a distributed collaborative video recorder that combines asynchronous file sharing with synchronous media streaming. DISCOVR extends our previous work

[1] A recent BitTorrent version includes trackless support. In practice, it makes every client a lightweight tracker using a Kademlia distributed hash table (DHT). This lowers the barrier to publish the content and eliminates a central point of failure. However, BitTorrent in trackless mode does not guarantee reliability. In trackless mode, the content publisher also loses the control of the content distribution: it cannot collect user statistics, and cannot stop the torrent and the distribution.

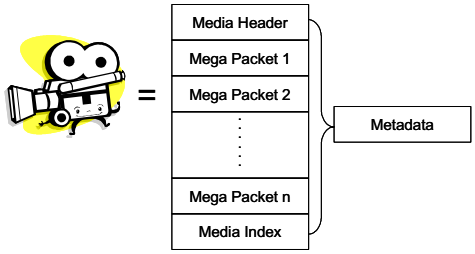

Figure 1 Media structure and metadata.

PeerStreaming, which is an on-demand P2P streaming system. In DISCOVR, the peers may switch between one of two modes: 1) in the asynchronous mode, the peer asynchronously shares and retrieves a list of media file for future viewing, and 2) in the synchronous streaming mode, the peer receives and plays back the media from arbitrary point designated by the user. The DISCOVR peers adopt a sender-driven priority based sharing protocol that ensures the request of the peers in both asynchronous and synchronous mode can be accommodated to the best interest of both groups of peers. By letting peers switch between streaming and asynchronous file sharing mode, the DISCOVR peers benefit by participating in the P2P activity longer, thus increases the available resources and capacity of the P2P network, and boosts the performance of the P2P streaming.

In the following part of the paper, we will discuss the components of the DISCOVR system in details. We discuss the media file format in Section 2. We show the system and network architecture in Section 3, and describe the sender-driven priority based sharing protocol in Section 4. The DISCOVR operation and experimental results are shown in Section 5. Finally, we give conclusions in Section 6.

2. Media File and Entity

The DISCOVR streaming media file, be it MPEG, AVI, windows media or real, can be segmented into media header, packets and media index, as shown in Figure 1. The media header contains global information of the media, e.g., the number of channels in the media, the property and characteristic (audio sampling rate, video resolution/frame rate) of each channel, media codecs used, author/copyright holder of the media, etc.. The body of the media consists of a sequence of media packets, each of which contains the compressed bitstream of a certain channel (audio/video) spanning across a short time period. Not all media packets are independently accessible. For example, in MPEG/Windows Media coded video, the user may start playing from any intra-coded frame, but not from any inter-coded frame. We thus cut the media file on group-of-pictures (GOP) boundaries, and group multiple audio/video packets into a mega packet, which is the smallest random access unit of the media. The media may have an optional index block, which converts access time to packet location so that the media can be accessed randomly.

In DISCOVR, we define the basic sharing unit as an entity. All DISCOVR entities share a flat ID space, e.g., a 128bit long word. In DISCOVR, the media header, index, and each mega media packet are all independent basic entities, each of which is a small file that can be cached on the disk, be shared in the network, and be accessed. Each basic entity may be further split into a number of fixed size blocks. The block is the smallest sending/receiving unit in the DISCOVR network. The entire media forms a directory entity. DISCOVR does not store the content for the directory entity, but forms a metadata file that records the compositing entities of the directory entity. The metadata file is an entity by itself and can be shared, accessed and cached in the DISCOVR system. Multiple media files sharing the same characteristics (e.g., a TV series) can be grouped into a bigger directory entity with the metadata describing the compositing entities as well. This way, DISCOVR may use the same protocol to share the media header, mega packets, media index and metadata file, and may easily identify the entity it shares.

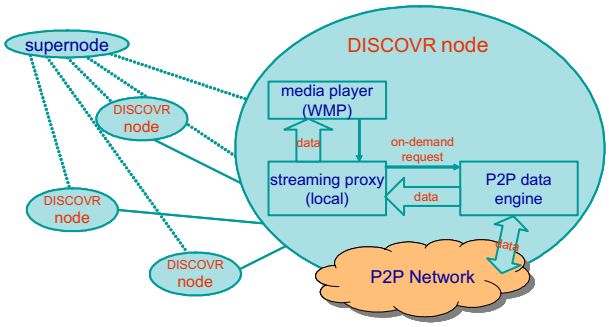

Figure 2 Distributed Collaborative Video Recorder (DISCOVR): system and network architecture.

3. System and Network Architecture

We show the system framework of DISCOVR in Figure 2. A DISCOVR peer consists of a P2P data engine and a streaming proxy. The streaming proxy implements a mini http-based media server on a local port (http://127.0.0.1:port/) that feeds data received by the P2P data engine to the media viewer, such as the windows media player. It also functions as the user interface of DISCOVR. The DISCOVR P2P data engine responds to both the asynchronous file sharing request and synchronous streaming request, and adjusts the operation mode of the DISCOVR peer accordingly.

The DISCOVR peer contributes both network bandwidth (for uploading content to the peers) and storage resource (for caching asynchronous shared media). The DISCOVR peer is connected to one or a cluster of supernodes, which track the list of entities shared by a certain DISCOVR peer. Since DISCOVR has two operation modes: asynchronous and synchronous, there are two lists for the shared entities as well. For the list of media files that are asynchronously shared for later viewing, the directory entities of the media file are placed in the asynchronous sharing list. For synchronous media streaming (e.g., viewing a media from a specific point), we place the entities of the mega packets that will be accessed for near future viewing on the synchronous sharing list, and place the directory entity of the media file again in the asynchronous sharing list. Thus, the on-demand media viewing peer are synchronously sharing the media packets for immediate viewing, and are asynchronously sharing the entire media file with the rest of the peers.

The DISCOVR supernode is designed to support a large number of peer nodes. Compared with the supernode in the FastTrack system, the DISCOVR supernode does not provide search functionality. Compared with the tracker node in the BitTorrent network, the DISCOVR supernode does not track the availability and/or the file sharing progress of the peers. Thus, most administrative traffic in the DISCOVR system goes directly through the peers.

4. DISCOVR: Peer Operations

4.1 Upon Connection

Shown in Figure 2, the DISCOVR uses an unstructured redundant network. Upon connection, the DISCOVR peer presents to the supernode the asynchronously shared and synchronously shared entity IDs. The supernode then points the DISCOVR peer to a number of random peers (say 20) that have commonly shared files, either synchronous or asynchronous, with the current peer. The DISCOVR peer may then attempt to establish connections with the list of the peers returned by the supernode.

Due to a variety of reasons, e.g., the presence of NAT/firewalls in the network, it is possible that not all connections between the peers can be established. In such a case, the DISCOVR peer may simply request for more new peers from the supernode. The DISCOVR peer may also terminate existing links and establish new links if its connected peers leave the network, or if there is no interesting entities from the connecting peers.

Let us now examine the operation after the peer connection is established. Let the two connected peers be *A* and *B*, respectively. Upon connection, the pair of peers first exchanges the list of asynchronous and synchronously shared entities. If peer *A* just starts sharing a media file, it will present the metadata that records the compositing entities of the media file as the sharing entity. Once the metadata is delivered, the directory entity of the entire media file will be listed as the sharing entity. Using the sharing entity list, both peers can quickly identify the entities that are commonly shared between the peers.

After the commonly shared entities are identified, the peers then exchange an availability vector that is a compact representation of what is held by the peer. For each entities (e.g., a media file), we first use a three-state tag to express whether the entity is fully available at the peer (tag '11'), is completely unavailable (tag '00'), or is partially available (tag '10'). If a directory entity is partially available, we further use the three-state tag ('11', '00' or '10') to identify if each of the compositing entities is fully available, completely unavailable, or partially available. If a basic entity is partially available, a bit vector is further attached to indicate what blocks of the entity is available.

4.2 Sender-driven priority based sharing protocol

It is the task of the DISCOVR P2P data engine to allocate the resource of the peers (the upload bandwidth) among the network links of the connected peers. Let us now consider a peer node *A*, which is connected to a set of peer nodes $\{B_1, B_2, \ldots\ldots, B_n\}$. Furthermore, let peer node B_i be connected to peer nodes $\{C_{i,1}, C_{i,2}, \ldots\ldots, C_{i,m}\}$. DISCOVR adopts a sender-driven priority based sharing protocol. For each link from peer node *A* to B_i, peer *A* determines what content to send in two steps: 1) throughput determination, 2) block determination.

For the first step, peer *A* will determine the proper throughput assigned to the link from *A* to B_i. We count the contribution of B_i to *A* as x_i, which is:

$$x_i = \alpha \cdot Th_{syn} + \beta \cdot Th_{async}, \quad (1)$$

where Th_{syn} is the throughput of synchronous access content from *A* to B_i, and Th_{asyn} is the throughput of asynchronous access content from *A* to B_i, α and β are weighting factors for the throughput allocation of synchronous and asynchronous content, respectively. We set $\alpha >> \beta$, so that wherever possible, the peer will first satisfy the synchronous access request, while the asynchronous file sharing takes a second priority. Let the upload bandwidth of peer *A* be up_A. The peer link from *A* to B_i will be assigned a throughput:

$$Th_i = \frac{x_i}{\sum x_i} up_A. \quad (2)$$

DISCOVR also implements an optimistic unchoking similar to that of BitTorrent[5]. The idea is to use the first available opportunity to send a random block to a newly joined peer so that the peer may quickly bootstrap itself. Afterwards, the throughput allocation in equ (2) is followed.

In the second step, peer *A* determines what blocks to send to peer B_i. First, peer *A* finds all sendable blocks to B_i. Those blocks belong to commonly shared entities of peers *A* and B_i, are held by peer A and are not delivered to peer B_i yet. We calculate a priority value for each of the sendable blocks. The higher the priority value, the more valuable that the block is to peer B_i. The priority value of a sendable block is the sum of the followings:

- receiver_on_demand [*default*=256]

Receiver_on_demand is added if the sendable block belongs to one of the synchronously accessed entities of peer B_i. The receiver_on_demand is a very large value, which ensures that synchronous accessing request is honored as much as possible.

- peer_on_demand_receiver [*default=16*]

Peer_on_demand_receiver is added if the sendable block belongs to one of the synchronously accessed entities of any connected peer $C_{i,j}$ of peer B_i. The idea is that this sendable block has a high value to peer $C_{i,j}$, and is thus of high resale value to peer B_i. If peer B_i are not operating in synchronous accessing mode, these sendable blocks will be sent first to ensure synchronous access of peer $C_{i,j}$ can be completed.

- Local_rarity [*default*=1]

Local_rarity is added for each sender's connected peer B_j ($j \neq i$) that is sharing but does not hold the block.

- Partial_entity_credit [*default*=4]

Partial_entity_credit is added for all the blocks belong to a partially available entity of peer B_j. The idea is to add an extra incentive to make the entity fully available.

- metadata_credit [*default*=16]

We give blocks of metadata entity extra priority for delivery.

4.3 Propose-to-send, confirm-to-receive and report-arrival-block

The sendable blocks are then sorted by priorities. Peer *A* then sends a propose-to-send (PTS) list of the highest priority blocks to peer B_i. Peer B_i examines the PTS list, checks if the blocks have already been proposed by the other peers, and compiles a confirm-to-receive (CTR) list and sends to peer *A*. The PTS request and the CTR confirm serve as a lock mechanism that ensures no duplicate block is sent. Because CTR list is an answer to the PTS request, CTR can simply be a bit mask of the block proposed by peer *A*. Only those blocks confirmed by the CTR message will be actually sent from peer *A* to B_i. At the same time that the confirmation CTR message is sent to peer *A*, peer B_i also compiles a report-arrival-block (RAB) message and sends to its other connected peers $\{C_{i,1}, C_{i,2}, \ldots\ldots, C_{i,m}\}$. Upon receiving the RAB messages, peer $C_{i,j}$ treats the RAB blocks as already held by peer B_i, and will not further propose the blocks. If for certain reason peer *A* crashes and is unable to actually send the blocks, peer B_i will send a negative RAB message to the peers $\{C_{i,1}, C_{i,2}, \ldots\ldots, C_{i,m}\}$ to re-allow the blocks in the RAB message to be proposed by peer $C_{i,j}$.

The process above describes how peer *A* may find blocks to send to its connected peer B_j. In similar process, peer B_j may find blocks to send to its connected peer *A* and $C_{i,j}$. Because it is the sender that first proposes the block, we say that DISCOVR adopts a sender-driven priority based sharing protocol.

5. DISCOVR Operation and Experimental Results

Figure 3 DISCOVR operation scene.

Table 1 Metadata of streamed media piece (samples).

Name	Size	Length	Bitrate	# of Entities
Dolphins	640x480	1:42	1.06mbps	21
Amazon	640x480	1:43	1.06mbps	23
Terminator	720x480	2:34:58	1.75mbps	1164
Die Hard	720x480	3:04:58	1.75mbps	1388

We demonstrate our DISCOVR system on a small P2P network consisting of 1 supernode and *n* peer nodes. We use *n*=5 for this specific experiment. A set of windows media movies is distributed through DISCOVR. The media includes short trailers, e.g., Delphins, and full length movies, e.g. Die Hard. We show the metadata of some sample movies in Table 1. Whenever a DISCOVR peer node is launched, it places all the media trailers in the asynchronous file sharing mode, and is downloading the media trailer for future viewing. The user may issue on-demand streaming requests to any of the peer node, for any media trailer (by clicking a URL of the trailer). An operation screen of the DISCOVR can be shown in Figure 3. When a user starts on-demand streaming of a media file, the media player sends an on-demand command to the mini http proxy, which converts the streaming request into a sequence of synchronous access requests of media packets that are sent to the underlying P2P data engine. The proxy implements an http-based streaming protocol and appears as a regular streaming server to the media player. It receives media header and packets from the P2P data engine, packs them conforming to the http streaming protocol and delivers to the media player on a real-time basis. It also supports VCR operations (e.g. seeking). The user may drag the progress bar in the media player. When that occurs, the media player will send a new http-seek request to the proxy, which translates the seek request to the synchronous access command of new media packets. The pending on-demand requests are cancelled, and new on-demand access requests are generated based on the new media playback position.

In Figure 4, we show the bandwidth allocation of a DISCOVR peer during the on-demand streaming process. In the figure, a DISCOVR client is synchronously streaming media from four other peers simultaneously. We show the throughput contribution of each of the four connected peers in Figure 4. We notice that each DISCOVR peer shares about $1/4^{th}$ of the serving load of the client. Moreover, we show the overhead of the DISCOVR protocol (the PTS, CTR, and RAB messages in comparison to actual data sent) in the top-most portion. We find that the overhead of the DISCOVR protocol is only 0.5% on average and thus almost negligible (and cannot be seen clearly in the figure) compared to data traffic. DISCOVR is a light-weight protocol that may support simultaneous asynchronous file sharing and synchronous media streaming.

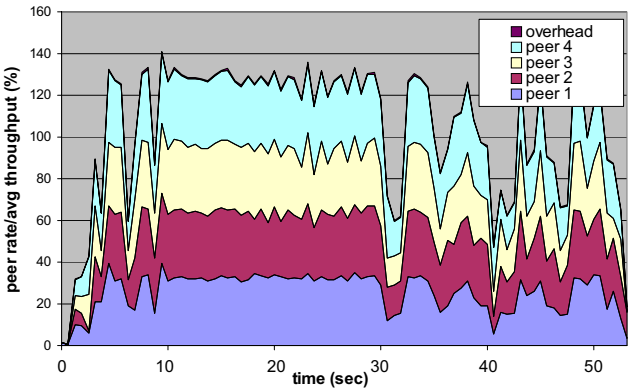

Figure 4 DISCOVR: peer throughput distribution.

6. Conclusions

We present DISCOVR, a P2P application that combines asynchronous file sharing with synchronous on-demand media streaming. DISCOVR uses a flat entity ID space for everything shared: media file, header, mega packets, index and metadata. All DISCOVR entities may be asynchronously or synchronously accessed. In DISCOVR, the peer nodes may flexibly switch between file sharing and media streaming modes, and use DISCOVR for both tasks.

7. References

[1] S. Saroiu, K. P. Gummadi and S. D. Gribble, "Measuring and analyzing the characteristics of Napster and Gnutella hosts", *in* Proc. 2003 Multimedia Systems, vol. 9, no. 2, Aug. 2003, pp. 170-184.

[2] H. Choon, N. Sarana and B. Rajkumar, "P2P networks for content sharing", Technical report, GRIDS-TR-2003-7, Univ. of Melbourne, Australia, Dec. 2003.

[3] A. Sharma, "The FastTrack Network", PC Quest magazine, Sept. 12, 2002.

[4] T. Hobfeld, K. Leibnitz, R. Pries, K. Tutschku, P. Tran-Gai, and K. Pawlikowski, "Information diffusion in eDonkey file sharing networks", Univ. of Wurzburg, Rep. No. 341, Sept. 2004.

[5] B. Cohen, "Incentives build robustness in BitTorrent", in *Proc.* Workshop on Economics of peer-to-peer systems, 2003.

[6] H. Deshpande, M. Bawa and H. Garcia-Molina, "Streaming Live Media over a Peer-to-Peer Network", *Stanford database group technical report (2001-20), Aug. 2001.*

[7] S. Banerjee, B. Bhattacharjee, C. Kommareddy, and G. Varghese, "Scalable application layer multicast", *Proc. Of ACM SIGCOMM'02*, pp. 205–220, Pittsburgh, PA, USA, August 2002.

[8] D. Tran, K. Hua, and T. Do. Zigzag, "ZIGZAG: An efficient peer-to-peer scheme for media streaming", In *Proc. of IEEE INFOCOM'03*, San Francisco, CA, USA, April 2003.

[9] CoolStreaming, http://www.coolstreaming.org.

[10] V. Padmanabhan and K. Sripanidkulchai, "The Case for Cooperative Networking", In *Proc. of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, Cambridge, MA, USA, March 2002.

[11] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth content distribution in a cooperative environment", In *Proc. of the International Workshop on Peer-to-Peer Systems*, Berkeley, CA, February, 2003.

[12] X. Jiang, Y. Dong, D. Xu, B. Bhargava, "GnuStream: a P2P Media streaming system prototype", In *Proc of IEEE Intern. Conf. on Multimedia and Expo (ICME 2003)*, Baltimore, MD, June 2003.

[13] J. Li, "PeerStreaming: A practical receiver-driven peer-to-peer media streaming system", in MSR-TR-2004-101. Sept. 2004.