CoolFish: A BitTorrent-Compatible P2P Video On-Demand and Live Streaming System Post-print

Authors: Li Zhezong, Zhang Tieying, Liu Yue, Cheng Xueqi

Date: 2017-03-09T00:00:00+00:00

Abstract

With the widespread adoption of online video services, peer-to-peer (P2P) transmission technology has garnered increasing attention from the industry. CoolFish, developed by us, is a streaming media system based on P2P technology that integrates both video on-demand and live streaming. In this paper, leveraging the CoolFish system, we present a detailed discussion and analysis of currently prevalent video transmission technologies, provide a comprehensive introduction to the architecture, functionality, and modular design of CoolFish, and furthermore, conduct an in-depth exploration of the key P2P transmission technologies and algorithms employed in the CoolFish system.

Full Text

CoolFish: A BitTorrent-Compatible P2P Video-on-Demand and Live Streaming System

Authors: Li Zhezhong, Zhang Tieying, Liu Yue, Cheng Xueqi

Abstract

With the widespread adoption of online video services, peer-to-peer (P2P) transmission technology has garnered increasing attention from industry. We have developed CoolFish, a streaming media system based on P2P technology that integrates both video-on-demand and live broadcasting. In this paper, we provide a detailed discussion and exposition of currently popular video transmission technologies based on the CoolFish system, present a comprehensive introduction to CoolFish' s architecture, functionality, and module design, and conduct an in-depth exploration of the key P2P technologies and algorithms employed in the CoolFish system.

Keywords: CoolFish, online video, streaming media, peer-to-peer (P2P), live broadcasting, video-on-demand, BitTorrent

1 Introduction

In recent years, network video services (streaming media) have become one of the most popular Internet applications. Online video refers to users watching video programs over the Internet. For instance, users can watch live television broadcasts directly online without a television set, or stream their favorite movies on demand without downloading entire files. Such convenient applications have naturally gained immense popularity among hundreds of millions of Internet users, leading to the continuous emergence of online video platforms like YouTube [1], Tudou [2], PPLive [3], UUSee [4], and PPStream [5].

The large-scale deployment of streaming media has been enabled by the rapid development of Internet hardware technology, particularly the swift improvement in end-user access bandwidth. Several years ago, Internet usage for home users was limited primarily to web browsing. However, with the significant alleviation of the "last-mile" bottleneck (where end-user access bandwidth was too low compared to backbone networks, resulting in poor user experience and underutilization of operator backbone bandwidth), an increasing number of users now choose to watch video programs online, enriching Internet applications. Yet, solving the last-mile problem has exacerbated the "first-mile" issue (where server-side upload bandwidth cannot meet user demand). Massive user data downloads frequently overwhelm media content servers. For example, YouTube, the largest video sharing service provider, pays millions of dollars monthly for data transmission [6] and must deploy super servers with 400 nodes and 10G high-speed networks to provide data transfer services.

To ensure such enormous download demands are met, industry and academia have proposed various video transmission technologies, including enhanced client/server models, Content Delivery Networks (CDN), peer-to-peer (P2P) transmission, and hybrid architectures combining multiple approaches. This paper provides a detailed exposition and comparison of these methods. Concurrently, we have designed, implemented, and deployed a P2P streaming media system—CoolFish. CoolFish is compatible with the BitTorrent network and provides both live broadcasting and video-on-demand services. This paper presents a comprehensive introduction to CoolFish' s architecture, functionality, and module design, and conducts an in-depth exploration of the key P2P technologies and algorithms involved in the CoolFish system.

2.1 Video Transmission Methods

Currently, video service providers employ two primary transmission methods: (1) **Download**, where received video content is stored as files on the user' s hard drive and can only be watched after completion; and (2) **Streaming media**, where users receive video content through online playback. This section

discusses these methods in detail.

2.1.1 Download

The download method stores received video content as files on the user' s hard drive, utilizing either centralized file servers or P2P networks for content distribution. Examples include Apple' s iTunes video service and Cinemanow' s Hollywood movie downloads. After downloading, files can be transferred to mobile video devices or legally shared within P2P networks.

2.2 Streaming Media

Streaming media provides users with a play-while-downloading experience, where video content is cached locally. Traditional streaming technology requires centralized streaming servers to distribute content. Today, video content can be distributed through P2P networks, significantly saving streaming server network traffic and maximizing concurrent user capacity under equivalent bandwidth conditions. For example, Microsoft Media Server Protocol (MMS) represents centralized streaming server technology, while PPLive and PPStream employ P2P streaming technology.

Streaming technology offers superior user experience by enabling continuous playback after downloading only partial media data. This approach does not store video files on the hard drive, requiring less storage space and accommodating devices with limited capacity such as PDAs (Personal Digital Assistants). However, streaming files cannot be transferred to other devices or the Internet. Streaming transmission builds upon mature multimedia encoding technology and network hardware/software development. To ensure continuous playback and satisfactory user experience, the download rate must at least exceed the program' s bitrate, making streaming technology more complex than download transmission.

Streaming services are divided into two categories: **live broadcasting** and **video-on-demand (VOD)**. Live broadcasting resembles television viewing—users join the system and watch current and future program content from the source. Video-on-demand resembles playing a video cassette—users can select what to watch, choose playback positions, and perform operations like pause, resume, fast-forward, rewind, and seeking.

2.3 Video Transmission Architectures

Due to the sensitivity of video services to bitrate, latency, and packet loss, video content transmission faces significant challenges in current packet-switched networks. With the explosive growth of Internet video applications, numerous architectures have been proposed to address these challenges. Popular architectures include client/server, enhanced client/server, CDN, P2P, and hybrid architectures.

2.3.1 Client/Server Architecture

The client/server architecture is the most traditional video transmission model. Clients connect directly to a central Web/FTP server to download video files or to a streaming server to obtain video streams, as shown in [Figure 1: see original paper].

2.3.2 Enhanced Client/Server Architecture

Unlike traditional client/server architectures that rely on a single video server, the enhanced client/server architecture allows clients to simultaneously obtain video content from multiple servers deployed across the Internet, as shown in [Figure 2: see original paper]. With multiple servers, this architecture effectively avoids single points of failure and enables load balancing among servers. Clients can also dynamically select servers based on current network conditions and latency information to achieve higher service quality.

2.3.3 Content Delivery Network (CDN)

CDN is a value-added network built upon existing IP infrastructure, deployed at the application layer. In CDN architecture, a group of CDN servers is deployed at the network edge, enabling users to obtain video content from nearby locations, thereby alleviating network congestion and reducing client waiting latency. The structure is shown in [Figure 3: see original paper]. CDN technology effectively improves bandwidth resource utilization, facilitating the popularization of streaming media applications on the Internet.

Traditional CDN technology can still be viewed as a client/server architecture. Although CDN distributes server capacity and video content across the network, improving streaming media service quality to some extent, its service scalability depends on CDN server node deployment, making service expansion extremely costly. Additionally, CDN architecture lacks elastic dynamic scaling capability, making it difficult to fundamentally improve overall system efficiency.

2.3.4 Peer-to-Peer Transmission

As a revolutionary technology in network architecture, P2P offers unparalleled advantages over traditional centralized services in performance and scalability across numerous application domains. Currently, P2P services have been deployed extensively, with their traffic constituting the majority of Internet traffic.

In P2P networks, resources and services are distributed across all nodes, with information transmission and service implementation occurring directly between nodes without server intervention, thus avoiding potential bottlenecks. The decentralized nature of P2P provides advantages in scalability and robustness. By leveraging substantial idle resources in the network, P2P can provide higher computing and storage capacity at lower cost. Since all participants can provide relay and forwarding functions, P2P significantly improves the flexibility

and reliability of anonymous communications, offering better privacy protection. Furthermore, as each node acts as both server and client in P2P networks, the requirements for server computing and storage capacity in traditional client/server architectures are reduced, and resource distribution across multiple nodes better achieves overall network load balancing.

Due to these characteristics, P2P architecture is widely applied in network video transmission, as seen in systems like CoolStreaming [8], PPLive [3], and PPStream [5].

2.3.5 Hybrid Architecture

Hybrid architecture combines multiple architectural models to integrate their respective advantages. Currently, the most popular combination merges CDN and P2P technologies, as exemplified by Kontiki' s hybrid video transmission solution. CDN-P2P fusion technology can be divided into two approaches:

1. **CDN-based transformation**: CDN server nodes use P2P technology for content distribution, exchange, and mutual backup, improving efficiency of content distribution from central servers to CDN servers.
2. **P2P-based enhancement**: Traditional P2P networks incorporate CDN management mechanisms and service capabilities, forming a video transmission architecture with CDN at the center and P2P at the edge. This integration provides stronger content and user manageability for P2P networks while making overall network traffic more manageable.

3.1 Functionality

CoolFish is a software system based on P2P streaming technology that integrates video-on-demand, live broadcasting, and local resource playback.

After downloading the CoolFish client, users can select from our latest released film resources, browse current popular content, and search for desired videos. Users can also download files using existing BitTorrent seeds and publish their own video resources. On the server side, dedicated log servers and Google Analytics-embedded Web servers collect and analyze user behavior statistics.

3.2 System Deployment

CoolFish is the first P2P streaming media system deployed on China Science and Technology Network (one of China' s four major ISPs: China Telecom, China Netcom, China Education and Research Network, and China Science and Technology Network). Current users are concentrated primarily on China Science and Technology Network and China Education and Research Network. Since deployment in October 2008, approximately 3.3 million users have accessed our system, with daily visits exceeding 7,000 and concurrent online users surpassing

700. [Figure 6: see original paper] shows user distribution in the CoolFish system. CoolFish supports high-definition movies up to 2.5 Mbps, with an average bitrate of 700 Kbps—more than 50% higher than the 450 Kbps typical of most P2P streaming systems [2,3,4].

In the CoolFish system, the server side deploys one tracker server, one program server, one seed server, one Web server, two super nodes, and one live broadcast resource publishing node.

1. Tracker Server: Similar to traditional BitTorrent systems, the tracker server collects and provides peer lists.

2. Program Server (Log Server): The program server collects program information from clients, which periodically connect to upload their lists of watched on-demand and live programs. The program server counts downloaders and publishers for each program, stores this information in a database, and requests clients to upload seed files when new tasks are detected, saving them locally.

3. Seed Server: The seed server receives seed files saved by the program server and provides seed download services to clients.

4. Web Server (Database): The Web server displays program statistics and popular program rankings from the database on the CoolFish official website and provides program search functionality. The client communicates with the website through its Web server communication module to retrieve program lists for display.

[Figure 7: see original paper] illustrates the CoolFish system deployment.

5. On-Demand Resource Publishing Node: This is essentially a regular CoolFish client. Administrators use the client' s publishing function to regularly update and release video resources. Super nodes function as "seeding" users in the BitTorrent network.

6. Live Broadcast Resource Publisher: The live broadcast publishing server introduces MMS into the P2P network to publish live programs. It uses the same live module as CoolFish clients, but for controllability of live program sources, the system currently does not allow ordinary CoolFish client users to publish live programs.

3.3 Module Structure

The CoolFish system comprises client and server components. The client is for general users to join the P2P network, watch on-demand and live programs, while the server serves global functions by presenting programs and collecting system statistics.

As a P2P application, the client is the system' s focus, with the following structure:

1. Apache Portable Runtime Libraries: The Apache Portable Runtime (APR) Libraries [9], developed by the Apache organization [10], are a cross-platform function library that encapsulates operating system APIs (including process/thread control, file system operations, network sockets, etc.) for Windows, Linux, OS/2, and other platforms. The interface provides C language format support, enabling source-code level compatibility across all APR-supported systems. Early Apache server programs handled platform-specific details internally for multiple platforms. As development progressed, the Apache organization extracted these functions into the APR library, which has been successfully applied in numerous open-source and commercial projects. CoolFish builds all components except the user interface and media player on the APR library foundation to facilitate future porting to Linux and other operating systems (the currently released CoolFish client is Windows-only).

2. P2P Core Module: The P2P Core module defines network data exchange patterns for upper-layer modules using object-oriented methods. Key classes include Connection, ConnectionPool, Engine, and Protocol. Upper-layer modules inherit these four classes and override their virtual functions. Protocol represents the module as a whole, with all external interactions implemented through its methods; Connection represents TCP connections established by the module (e.g., in the on-demand module, Connection subclasses represent connections between nodes for exchanging data chunks); all Connections for a module are centrally managed in its ConnectionPool; Engine manages three threads: Read, Write, and Monitor. The first two handle reading and writing for all Connections in the module' s ConnectionPool, while Monitor manages Connections (e.g., in the on-demand module, periodically statistics like download speeds are collected, and disconnected connections are removed). Thus, development of upper-layer modules follows the P2P Core pattern, requiring only implementation of content processing for read/write operations and connection management. This design simplifies module development, provides a unified pattern, and facilitates system extension.

3. Tracker Communication Module: This module communicates with tracker servers to obtain peer lists. The tracker server is a centralized server introduced by the BitTorrent protocol. After joining, nodes must register with the tracker specified in the download task, announcing their participation, and obtain peer lists from the tracker to establish connections. Since CoolFish' s on-demand module is BitTorrent-compatible, tracker communication is necessary. Additionally, the live module uses the same method for peer list acquisition. The module' s function is: the caller specifies a task, provides either a BT seed for on-demand programs or a hash code for live programs, and the module periodically connects to the tracker to obtain peer lists for the caller' s use.

4. Program Server Communication Module: This module informs the CoolFish program server about the on-demand and live programs being downloaded or watched by the local node.

5. Web Server Communication Module: This module periodically con-

nects to the CoolFish official website to retrieve program lists and other information for display on the client.

6. On-Demand Module: The on-demand module comprises two sub-modules: data transmission and file management. The data transmission module establishes connections with other nodes, uploads and downloads data chunks, and submits them to the file management module. The file management module receives downloaded data, writes it to media files stored on the local hard drive, and provides chunks to the data transmission module when needed. Meanwhile, the media player reads media files through shared access with the file management module. Consequently, the file management module must frequently respond to queries from the client main program about whether certain data segments have been completely written and are ready for playback, and upon learning the playback position, should inform the data transmission module to schedule downloads of required chunks accordingly.

The on-demand module's protocol is BitTorrent-compatible to leverage resources outside the CoolFish system. Thus, users can open seed files downloaded from other BT sites to join the download network. For popular seeds, satisfactory on-demand effects can be achieved. However, traditional BitTorrent protocol cannot be directly applied to streaming media on-demand for three reasons: First, BitTorrent uses a rarest-first chunk selection strategy, which, while beneficial for file sharing by preserving rare chunks, results in random chunk arrival order, making it difficult to quickly obtain contiguous data for playback. Second, BitTorrent employs a tit-for-tat strategy to encourage uploading, but the resulting frequent choking increases network dynamics, making it difficult to meet streaming media' s stability requirements. Third, BitTorrent protocol does not specify detailed rules for chunk scheduling and download task allocation, necessitating a faster chunk scheduling scheme for on-demand systems.

7. Live Module: The live module enables nodes to join live broadcast P2P networks, exchange media data chunks, and publish and watch programs. As a live program publisher, a node must connect to an MMS server to introduce the live data stream into the P2P live system. This server-client approach requires good connection performance (typically the MMS server and live publishing node reside on the same host). This represents the initial data source for the live network. The live module design allows multiple live publishing nodes to connect to a single MMS service. Since they calculate identical hash codes based on the MMS service, they actually form a unified live network rather than being sole publishers of separate networks. This creates larger-scale live networks that leverage P2P advantages, distributes pressure from single publishing nodes, and avoids unnecessary duplication.

As a live program viewer, the live module, besides downloading and transmitting data chunks, launches a local MMS service. Downloaded chunks are sequentially submitted to the local MMS server, and the media player connects directly to this local MMS server port to watch programs. This makes interaction between the client main program and live module very simple: merely inform what

program to watch and have the player connect to the local MMS server.

4.1.1 BitTorrent Protocol Overview

BitTorrent (BT) is a file distribution protocol that has become the most popular collaborative file-sharing system on the Internet. BitTorrent shares identical files among organized system nodes, rapidly and efficiently distributing and replicating the shared file to all other nodes. BT leverages each node' s bandwidth to effectively copy shared files to numerous requesting nodes, aiming to quickly and efficiently distribute and download large files. A fundamental concept in BitTorrent is dividing a shared file into many equal-sized pieces (typically 256KB). A participating node can download different pieces in parallel from multiple nodes while simultaneously uploading owned pieces to other requesting nodes. BitTorrent uses SHA1 hashing to ensure successful reassembly of all downloaded pieces. Additionally, data piece exchange between nodes is driven by a tit-for-tat incentive mechanism.

Key algorithms used in the BitTorrent protocol are introduced below.

1. Choking and Optimistic Unchoking Algorithms: BitTorrent employs choking algorithms for multiple reasons, primarily to implement tit-for-tat incentive mechanisms between nodes, ensuring stable download rates and increasing sharing probability among users.

Each BitTorrent node maintains unchoked connections with a fixed number of peers. Under the tit-for-tat mechanism, each node uploads data pieces to its neighbors, strictly determining which nodes should remain unchoked and which should be denied upload services based on current download rates, thus establishing its connection set. Denying upload service is called "choking," and the corresponding algorithm is the choking algorithm.

BitTorrent also employs optimistic unchoking to select nodes for uploading. In optimistic unchoking, besides selecting a fixed number of nodes for uploading based on tit-for-tat strategy, each participating node randomly selects one requesting node for uploading without considering that node' s past upload rate. Thus, at any time, each node maintains one optimistically unchoked connection that remains unblocked regardless of its download rate. Optimistic unchoking serves two main purposes: first, nodes can try connecting to previously unconnected peers to potentially find better connections; second, when new nodes join without any file data contribution, optimistic unchoking ensures they can quickly download some file content and begin participating in data exchange. Optimistic unchoking rotates the selected node every 30 seconds.

2. Rarest First: Rarest First means a downloading node selects the target piece that is owned by the fewest neighbors—i.e., the most scarce piece—for download. This strategy is called the "Rarest First strategy."Under this strategy, the entire system tends toward an optimized state where data pieces are evenly distributed, improving overall system performance, preventing disruption from

nodes with rare pieces leaving early, and enhancing system stability.

4.1.2 CoolFish Modifications to the BitTorrent Protocol

In CoolFish' s on-demand module, we modified traditional BitTorrent protocol to enable CoolFish clients to exchange data with general BT clients while better meeting on-demand requirements. First, we replaced "rarest-first" with "most-needed-first" chunk selection. Second, we use a cross-linked list to manage download tasks. Third, for upload data management, we introduced priorities for each neighbor node Pj, prioritizing upload requests from higher-priority nodes. The priority adjustment method based on speed is similar to BitTorrent' s tit-for-tat strategy—preferentially serving nodes that provide high-quality service to encourage contribution—but employs a smoother decision mechanism than tit-for-tat' s "choke-unchoke" approach, better adapting to streaming media' s need for stable data flows.

4.2 Multi-Task Downloading

In typical video-on-demand systems, users can only download and share the program they are currently watching, a strategy we call Single-Task Caching (STC). Since CoolFish is fully BitTorrent-compatible, it provides Multi-Task Caching (MTC) technology similar to ordinary BT clients, allowing users to download and share other programs while watching current content. This technology improves video sharing rates and upload speeds to some extent, as shown in [Figure 9: see original paper].

4.2.1 Mediacoop

Mediacoop is a structured query algorithm for P2P video-on-demand that combines content and quality matching. Mediacoop provides query services based on playback point distance and global network latency to reduce client startup delay and seeking delay while improving playback continuity. The Mediacoop query process consists of two steps. First is content matching: based on neighbors' playback point information in routing tables, recursive queries locate a group of nearby playback points containing the required content, as shown in [FIGURE:10(a)]. Second, based on the global network latency table, nodes with low latency are selected for connection, as shown in [FIGURE:10(b)].

4.3 Live Streaming Data Exchange Strategy

The current CoolFish live module employs a classic "pull" strategy. Nodes periodically broadcast buffer piece possession status (PieceMap). The buffer is divided into several windows, which are the units for data submission to the local MMS server. Nodes allocate download tasks based on required pieces' distribution among neighbors and each neighbor' s capacity. For each piece request, a waiting time is estimated. If a piece fails to arrive within the waiting

time, a "first-timeout mechanism" is triggered, requesting the piece from all other neighbors possessing it. If the piece still fails to arrive on time, this indicates the node' s required data has fallen far behind the network' s playback progress, rendering data obsolete, or that massive neighbor disconnections have occurred. In this case, a "second-timeout mechanism" is activated, potentially re-establishing connections with neighbors and re-determining the buffer' s starting position based on neighbors' PieceMap.

During piece downloading, if the foremost window is completely filled, it can be submitted to the local MMS server and the buffer shifted forward. Sometimes, the foremost window may have only a few pieces missing. Continuing to wait or activating the first-timeout mechanism would slow buffer advancement, easily causing the node to fall behind other nodes and trigger second-timeout. Since media players handle occasional missing pieces well (or can substitute empty data pieces), the window can be directly submitted to the local MMS server and the buffer shifted forward, preventing individual missing pieces from affecting overall download progress.

5 Conclusion

In recent years, P2P network applications have become increasingly widespread, with research continuously deepening. Streaming media represents a crucial application domain for P2P networks. Research on P2P streaming media has undergone years of development, leveraging the collaborative nature of participating nodes to achieve significant progress, with numerous successful commercial systems emerging.

CoolFish is a software system based on P2P streaming technology that integrates video-on-demand, live broadcasting, and local resource playback. Currently, the system has been in service for a short period, leaving several meaningful tasks for future work, such as providing personalized services based on user behavior analysis, adding distributed search functionality, and implementing multi-torrent multi-protocol interoperability to further improve download speeds.

Note: Figure translations are in progress. See original paper for figures.