HYBRID VIDEO DOWNLOADING/STREAMING OVER PEER-TO-PEER NETWORKS[1]

Yufeng Shan and Shivkumar Kalyanaraman

Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute, Troy NY 12180
{shany, kalyas}@rpi.edu

ABSTRACT

Peer-to-peer based multimedia delivery is becoming increasingly more important in today's networks. Using a peer-to-peer network to assist video streaming is a topic of considerable interest. In this paper, we propose a novel hybrid video downloading/streaming scheme (HDS) that efficiently *integrates* traditional client/server based video streaming and peer-to-peer based media distribution. Furthermore, we propose a receiver-driven algorithm to coordinate the downloading and streaming modes; and control the state transitions between these modes. We have performed real-world experiments and simulations to validate our concept. These results show that our proposed scheme greatly increases the availability of video content on the receiver side and simultaneously reduces the server load significantly.

1. INTRODUCTION

High-quality video streaming over the current best-effort Internet is a challenging proposition due to video requirements such as high bit rate requirement, delay and loss sensitivity. Streaming media distribution has been an intensively studied research topic in the past several years. In the area of source coding, methods such as layered coding, error-resilience coding and Multiple Description Coding (MDC) have been proposed. In the area of channel coding, Forward Error Correction (FEC) techniques have been proposed to combat the channel losses while reducing delay due to retransmission. In the area of network architecture, companies such as Akamai and Digital Island have deployed Content Delivery Networks (CDNs) by using a network edge-based architecture (edge servers) to achieve load balancing, lower latency and higher through-put. Most recently, peer-to-peer (P2P) architectures are gaining attention. In this model, a peer stores the streamed data after receiving it, and then streams the cached content to other requesting peers. For example, Padmanabhan et al [1] propose a solution called CoopNet, an approach for content distribution that combines aspects of infrastructure-based and peer-to-peer based content distribution, where client cooperate to distribute content, thereby alleviating the load on the server. CoopNet builds multiple distribution trees spanning the source and all the receivers for it MDC coded media content. Xu et al [2] propose an optimal media data assignment algorithm to assign media data to multiple peers in one streaming session and a distributed differentiated admission control protocol to quickly amplify the system's total streaming capacity. Yeo et al [3] propose an application level multicast overlay using peering technology and a lightweight gossip mechanism to monitor prevailing network conditions and to improve the tree robustness. In [4] the author designs a peer-to-peer technique called ZIGZAG for single source media streaming. ZIGZAG allows the media server to distribute content to many clients by organizing them into an appropriate multicast tree rooted at the server.

Most of the papers talk about massive video data distribution using application layer *multicast* based on peer-to-peer overlay. In [5], the authors measure two typical peer-to-peer networks, Napster and Gnutella, according to the characteristics of the participating hosts such as reported Internet connection speed, latencies, lifetimes, shared data and so on. Their results show the peer-to-peer network is heterogeneous and dynamic; only *less than 5% hosts can work as server-like peers*. Thus, we argue that there are problems in a peer-to-peer based video streaming architecture. First, it should have sufficient number of *powerful peers* (in terms of computation, bandwidth, memory and disk capacity) with cached video data at the beginning of the streaming session. Second, due to the *dynamic and heterogeneous characteristics of peers*, the clients may suffer more network fluctuation and network outage than the traditional client/server structure. Third, in extreme case, the streaming session has to be closed when all the peers with cached content are unreachable. On the other hand, the traditional video server is always *available*.

Based on the above arguments, in this paper, we propose a novel *hybrid* video downloading/streaming scheme (HDS) that efficiently *integrates* traditional client/server based video streaming system (streaming mode) and peer-to-peer based media data distribution system (downloading mode). In our hybrid architecture, the two modes *complement* each other. Furthermore, we propose a receiver-driven coordination control algorithm (RDCC) to coordinate downloading & streaming modes; and control the state transition between these modes.

The major contributions of this paper are: (1) our proposed HDS scheme efficiently integrates traditional client/server based streaming with peer-to-peer based media distribution to *significantly reduce the server load*. (2) The proposed receiver-driven algorithm maintains the *maximum content availability* at receiver side by leveraging both the streaming and downloading modes. (3) Given all peers with cached content are unavailable,

[1] This paper is sponsored by ARO grant DAAD19-00-1-0559 and supported in part by a grant from Intel Corp.

the receiver still can maintain video streaming from video server.

This paper is organized as following: The proposed hybrid video downloading/streaming scheme is discussed at Section 2; in Section 3, a receiver-driven coordination control algorithm is discussed; A memory disk cooperative buffering scheme is discuss in Section 4; Experiments and discussions are present in Section 5; followed by conclusions in Section 6.

2. HDS SCHEME

Our proposed hybrid video downloading/streaming (HDS) architecture is show as Figure 1. In order to simplify the description, we outline our scheme with one video server, one "*supplying*" peer (i.e. with cached content), one "*requesting*" peer and a CBR video sequence.

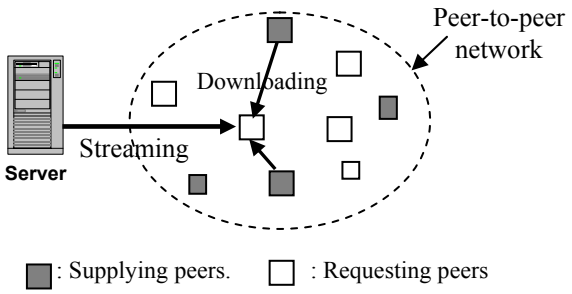

Figure1: Hybrid video downloading/streaming architecture

The building blocks of our proposed HDS system include (1) receiver-driven coordination control scheme (RDCC) and memory disk cooperative buffering (MDB) scheme at requesting peer; (2) video server scheduler; (3) supplying peer scheduler. All these building blocks efficiently cooperate to reduce the server load and maximize the availability of video content in receiver side.

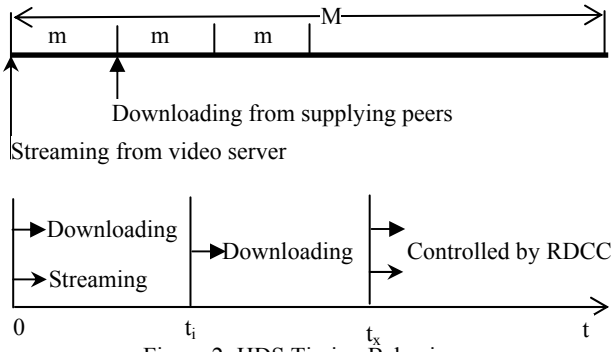

Figure 2: HDS Timing Behavior

Whenever a peer decides to watch a movie, first it sends out a request to the video server. The video server can be a traditional video server or a CDN video server. In the meantime, the requesting peer performs content lookup service in a peer-to-peer network, such as Chord [6], to find the supplying peers who have the requested video content. After the receiver gets the video profile from the server and the addresses of the supplying peers from the network, it analyzes the video content according to the video profile, divides the whole video content into *N* slices as Figure 2 (upper subfigure). We define a "*slice*" as a piece of video data in a video bit stream; one slice may include several video frames. Suppose the total amount of requested video content is *M* bytes, the corresponding bytes for each slice $m = M/N$ and the display time for each slice is t_i. The receiver runs the proposed RDCC algorithm to coordinate downloading and streaming modes described in Figure 2 (bottom subfigure).

1. At time 0, the receiver starts to receive streaming traffic (the beginning of the video content) from the video server and pre-buffers up to T_{pre} seconds using an in-memory playout buffer. *Simultaneously*, the receiver *also* starts to download video content from a supplying peer. But this content starts at the *second slice*, i.e. at a staggered position in the overall video content sequence.
2. At time t_i, the streaming session catches up to the position of the second slice in the video content. The server scheduler now suspends the streaming mode, thus offering relief to the server. The working mode is now the downloading mode from supplying peer, i.e. peer-based download-only.
3. During the download-only stage, the receiver analyzes the availability of the video content in the receiver side starting from time t_i. At certain time t_x, when the amount of data in receiver buffer is lower than a buffer threshold (indicating an increased probability of buffer underflow), then the RDCC fashions an optimum mix of server-based streaming and peer-based download. The overall objective is to shield the user from variations in network bandwidth, server/peer overload, peer-transience etc and maximize availability of the video stream *as perceived by the user*. The RDCC algorithm controls and coordinates these modes according to the available bandwidth and the availability of video content in receiver buffer. We discuss the scheme in detail in Section 3 and Section 4.

3. RECEIVER-DRIVEN COORDINATION CONTROL ALGORITHM

The RDCC algorithm is the key part of the overall HDS scheme. It computes the availability of the video content in receiver buffer, estimates the available bandwidth and coordinates the download-ing and streaming modes.

3.1 Availability of the video content

We define the availability of video content *r* as the ratio of total successive data in the buffer to the pre-buffer size as Equation (1). For example: if the receiver is displaying the *n*th slice of video data, then the *(n+1)*th video slice in receiver buffer is called "*successive*" data. The *(n+3)*th video slice is not called successive data *until* the *(n+2)*th video slice is buffered in the receiver side. In the following equation,

$$r = \frac{T_{total}}{T_{pre}} \quad (1)$$

r is the availability of video content in the receiver buffer; T_{pre} is the pre-buffer size, T_{total} is the total amount of successive data in receiver buffer.

This *r* is used by RDCC to measure the receiver buffer conditions, if $r > 1$, the receiver buffer has enough successive data, if $r < 1$, the receiver buffer suffers a certain underflow.

3.2 RDCC algorithm

The goal of the receiver-driven coordination control algorithm (RDCC) is to minimize the server load and to maximize the availability of video content in receiver buffer. The receiver measures the available bandwidth B_s between the video server and receiver, the available bandwidth B_d between the "supplying" peer and the receiver using a TCP-friendly algorithm [7]. We define the bit rate for maintaining a smooth video display as B. The RDCC algorithm is described as following.

```
If (r<1 && Bd <B) {
  Trigger the streaming mode;
  If (Bs >=B) {
     Downloading (n+1)th video slice given current slice
     being streamed is nth;
     if (streaming session reaches (n+1)th video slice)
        Suspend streaming mode;
  } else
      run COOP mode;
} else
  downloading-only mode;
```

In COOP mode, the server and supplying peer cooperate to maintain the availability of the video content in the receiver buffer. This COOP mode only happens when the calculated availability of video content r is less than 1 and both B_s and B_d are smaller than B. In COOP mode, the receiver calculates the ratio R_g between the available bandwidth of video server and the available bandwidth of the supplying peer, as in Equation (2) and then sends the ratio result back to both server and supplying peer.

$$R_g = \frac{B_s}{B_d} \tag{2}$$

Based on one GOP, both the video server and supplying peer schedulers calculate the amount of frames they should send according to the ratio R_g, and send the video frames as Figure 3. In HDS system, the default protocol for the schedulers is that the video server sends the first parts of the GOP and the supplying peer sends the remaining parts of the GOP, as shown in Figure (3), the video server transmits the black parts and the supplying peer transmits the grey parts in one GOP according to R_g.

Figure 3: The HDS scheduler

4. MEMORY DISK COOPERATIVE BUFFERING

Traditional video streaming system uses a limited *in-memory playout buffering* scheme to absorb the network bandwidth fluctuations. When the network conditions become bad and the display buffer underflows, the playout process is stopped and waits for some data to be buffered. We refer to this action as "stop-and-re-buffering" and would like to minimize such instances because the user sees stalled video during this time. On the other hand, the receiver *cannot buffer* more data than the predefined in-memory buffer size, *even if the available bandwidth is much larger than video bit rate*. In this paper, we propose a *memory-disk* cooperative buffer scheme (MDB) that can supply virtually unlimited buffer capacity and can efficiently use the available network bandwidth. The structure of the buffer scheme is described as Figure 4.

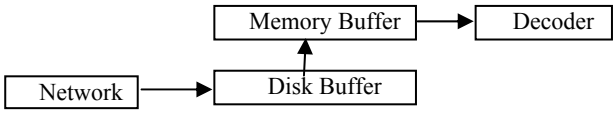

Figure 4: Memory Disk Cooperative Buffering

The MDB scheme uses two buffers, one memory buffer and one disk buffer. The size of the memory buffer is the size of predefined buffer. The disk buffer is almost unlimited size compared to a movie length. Whenever a packet comes from the network, the MDB scheme first caches the packet into disk buffer, and then fills the memory buffer. There are advantages using MDB scheme. (1) It can absorb more network bandwidth fluctuation, especially in a dynamic peer-to-peer network. With a play-out buffer that significantly longer than a round trip time, our system behaves very similar to an erasure channel with an unlimited number of retransmissions allowable for each packet. (2) Significantly reduce the stop-and-re-buffering instances. (3) It can efficiently use the available bandwidth, especially in downloading mode; the receiver can use up as much bandwidth as available to buffer video data.

5 EXPERIMENTS AND DISCUSSIONS

We setup a test bed as shown in Figure 5 to test our schemes. The receiver is a laptop in the RPI university network, two computers at RPI network lab act as supplying peers, the video server is a SUN machine at UC Berkeley. The bandwidth is controlled by a PC running NISTNet[8]. Test video sequence is "foreman", QCIF format, H.263+ CBR encoded, average bit rate is at *B=128kbps*. The length of the sequence is 81 seconds.

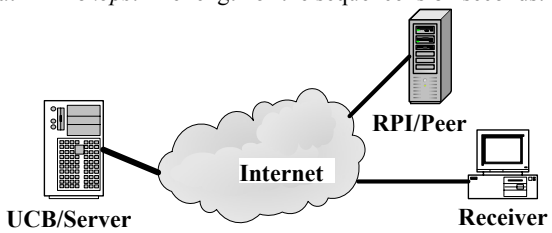

Figure5: The Topology of our test bed

5.1 Memory Disk Cooperative Buffering

The MDB scheme efficiently uses disk to offer nearly unlimited buffering capacity for video streaming. It can use as much bandwidth as available. In this section we compare our MDB scheme with traditional Memory-Only-Buffering scheme (MOB). The pre-buffering time for both schemes is 3 seconds. The NISTnet tool is used to control the available bandwidth as indicated in Figure 6(a). Figure 6 (b) shows the receiving rate of MOB and MDB scheme. Traditional MOB scheme cannot use more available bandwidth due to the limited memory buffer size. On the other hand, our MDB scheme can use as much bandwidth as available to cache video data. Thus, the amount of buffered data in MDB scheme buffer is much more than the data in MOB

scheme buffer when the available bandwidth is larger than the video bit rate as show in Figure 6(d). During the network bandwidth fluctuates, the MDB scheme can absorb more fluctuation than MOB does, so there are much less stop-and-re-buffering instances of MDB scheme compared with MOB scheme as show at Figure 6 (c).

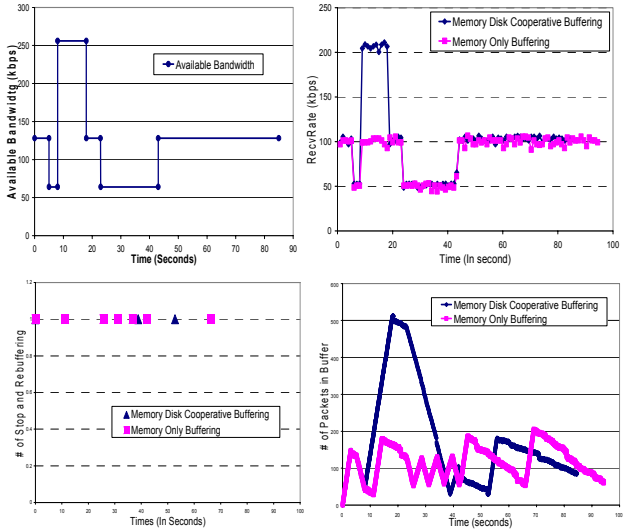

Figure 6: (a) top left, available bandwidth; (b) top right, receiving rate; (c) bottom left, stop-and-re-buffering times; (d) buffer occupation.

5.2 RDCC algorithm.

RDCC algorithm significantly reduces the server load and in the meantime maximizes the availability of video content by leveraging the streaming mode and downloading mode. In this section, we define the server load as "1" if the server sends video at 128kbps and as "0.5" at 64kbps.

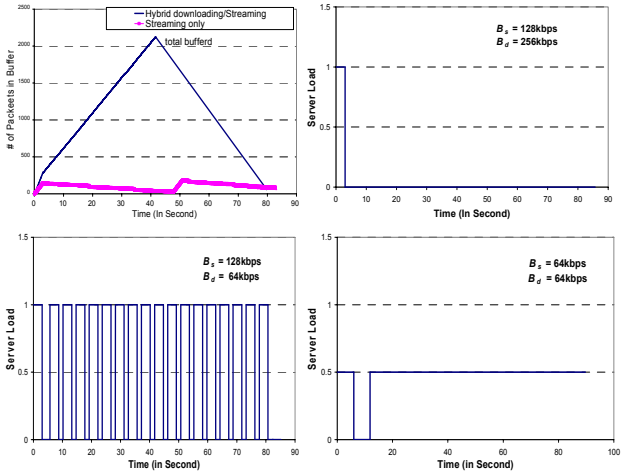

Figure 7: (a) top left, buffer occupation; (b) top right, (c) bottom left, (d) bottom right: server load in different available bandwidth

We set $T_{pre}=t_i=3$ seconds. Figure 7(a) shows the buffered video data in the situation of B_s=128kbps and B_d=256kbps. Where B_s is the available bandwidth between video server and receiver; B_d is the available bandwidth between supplying peer and receiver. Because the B_d is larger than video bit rate B, after the first stage of streaming, the RDCC controls the working mode at downloading-only mode according to the calculated availability of video content r. The MDB buffering scheme tries to use as much bandwidth as it can to buffer video data. In Figure 7 (a), all the video data is buffered in the receiver buffer at around 41 seconds. The video server only works at the pre-buffering time as Figure 7(b). On the other hand, the compared traditional streaming scheme maintains a lower buffer occupation, the video server need to work during the whole session, 84 seconds. In Figure 7 (c), B_s=128kbps and B_d=64kbps, RDCC controls the streaming mode to work periodically and the downloading mode assists the streaming mode. In Figure 7 (d), B_s=64kbps and B_d=64kbps, neither of the available bandwidth is equal or larger than video bit rate B. The RDCC works at COOP mode. The streaming mode and downloading mode cooperate to maintain the video displaying. The server-load is half of the traditional streaming system, but the receiver maintains a better video quality.

6. CONCLUSSIONS

In this paper, we have proposed a novel hybrid video downloading/streaming scheme (HDS) that efficiently integrates traditional client/server based video streaming system and peer-to-peer based media data distribution system. Furthermore, we propose a receiver-driven algorithm to coordinate downloading & streaming mode and control the state transit between downloading mode and streaming mode. We have demonstrated the effectiveness of the HDS scheme. It significantly reduces the server load and increases the availability of the video content. Future work will focus on cooperative caching scheme and multimedia transport protocols over peer-to-peer network.

7. REFERENCES

[1] Venkata N. Padmanabhan, Helen J. Wang, Philip A. Chou Kunwadee Sripanidkulchai "Distributing Streaming Media Content Using Cooperative Networking" Microsoft technical report MSR-TR-202-37, April 2002

[2] Dongyan Xu, Mohamed Hefeeda, Susanne Hambrusch, Bharat Bhargava "On Peer-to-Peer Media Streaming", Purdue Computer Science Technical Report, Apr. 2002

[3] C.K. Yeo, B.S.Lee and M.H.Er "A Peering Architecture for Ubiquitous IP Multicast Streaming" ACM SIGOPS Operating Systems Review, Volume 36, Issue 3, pp 82-95, July 2002

[4] Duc A. Tran Kien A. Hua Tai T. Do "ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming" University of Central Florida Technical Report 2002.

[5] Stefan Saroiu, P. Krishna Gummadi, Steven D. Gribble "A Measurement Study of Peer-to-Peer File Sharing Systems" Tech Report # UW-CSE-01-06-02 University of Washington

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. *In Proceedings of SIGCOMM '01*, pages 149-160, San Diego, California, August 2001.

[7] S. Floyd, M. Handley, J. Padhye, and J. Widmer. "Equation-Based Congestion Control for Unicast Applications" *Proc. ACM SIGCOMM*, pages 43--54, September 2000.

[8] http://snad.ncsl.nist.gov/itg/nistnet.