A Packet-Based Caching Proxy with Loss Recovery for Video Streaming

Kuan-Sheng Hsueh and Sheng-De Wang
Department of Electrical Engineering, National Taiwan University
{kshsueh, sdwang}@hpc.ee.ntu.edu.tw

Abstract

With the popularity of broadband networks, video streaming grows rapidly in the Internet. By the deployment of caching proxies, the backbone bandwidth can be saved significantly. In this paper, we propose a packet-based caching architecture for video streaming. The proposed caching scheme is based the streamed packets instead of video files and the consideration of packet loss recovery. We also propose an effective cache replacement algorithm PLFU and evaluate the algorithm through simulation.

1. Introduction

With the rapid growing of broadband networks, audio and video programs become more popular in the Internet. Using the streaming technique, the client can play the media via coming packets in real time. Video streaming is sensitive to loss and delay, so it requires a stable network bandwidth. The current Internet transmits packets in best-effort way and can't guarantee the QoS(quality of service) for video streaming. Most video streaming applications use UDP as their underlying protocol. However, lots of UDP packets transmitted without congestion control will disturb other TCP connections and jam the network. It is important to find a feasible solution for the explosively increasing video applications.

Caching proxies widely used in WWW have good performance in solving network congestion and server overloading. The caching proxy for video streaming is a solution to the problems for current Internet architecture. By caching the popular video clips in the caching proxy, the backbone bandwidth can be saved greatly. Because of the good QoS of local area network (LAN), clients can also receive higher video quality from the caching proxy. For real time program, caching proxies can receive unicast packets from servers and multicast these packets to the clients in LAN.

The caching scheme of video streaming is quite different from traditional file caching. First, the size and accessing duration of a video file is longer than a html or an image file, and clients may play part of it only. Second, packets of a video streaming session are transmitted in real time. A caching proxy must have the ability to intercept these packets and cache them in files while forwarding these packets to the clients. If there are requests for the same video in the next time, the caching proxy can stream packets to clients from the cache directly. Therefore, it's beneficial to have an appropriate cache file format to speed up the processing. Third, during the streaming session, UDP packets may suffer packet loss, and a caching proxy should have the responsibility to repair the loss for preventing error propagation. Additionally, the thinking of the cache replacement algorithm is not the same as the object-based either.

In this paper, we propose a packet-based caching proxy scheme with packet loss recovery. By caching the RTP packets of the video streams in special file format, the caching proxy can access every packet easily. Traditional video caching proxies, which use original video format as cache file, incur overhead of partitioning video file to packets for every requests, and it is not easy to repair lost packets. We can describe the proposed caching scheme in four steps, and implement the caching proxy in a module-based design. With the repairing step, the proposed proxy scheme can maintain loss-free caching.

The resource of a caching proxy is limited. When the cache space is full, some unused data must be removed. The caching length of a video clip is based on the access frequency and playing time. We proposed a replacement algorithm called partial least frequently used (PLFU) to utilize the cache space more efficiently. At last, we evaluate our algorithm through simulations.

Our design can make hierarchical cache, and the caching process is transparent to clients. By using the proposed packet-based cache file format, we can reduce the processing overhead of caching proxies.

The rest of this paper is organized as follows. In section 2, we address related work of the video streaming and caching proxy. Section 3 describes our design and the caching scheme. We present our implementation details in section 4. We proposed our cache replacement algorithm in section 5 and evaluate the algorithm in section 6. Finally, section 7 concludes the paper and discusses our future work.

Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02)
0-7695-1852-4/02 $17.00 © 2002 IEEE

2. Related Work

Video Streaming through multicast in the Internet can reduce the usage of the network bandwidth. Eager et al. [1] proposed a batching scheme to serve the same requests by multicast during a short period, and Hua et al. [2] proposed another multicast idea. However, multicasting over the Internet backbone is widely limited. Also, packets need to be sent again when a new request occurs in the next time. Zhang et al. [3] developed a caching scheme called video staging. By caching a part of video in the proxy, the backbone bandwidth can be saved. In [4], prefix caching is proposed to reduce the play latency. Kangasharju et al. [5] and Rejaie et al. [6] cached the layered encoded video and proposed their replacement algorithm. A popularity-based partial caching scheme with the SCU replacement policy is presented in [7-8].

Hilt et al. [9] used ACK and NACK records to repair the lost packets after streaming. Zink et al. [10] developed a Loss-Collection RTP Protocol to transmit reliable video files to caching proxies. Some of the above existing work didn't mention their video cache format, and some used original file format. This paper is focused on designing a loss-free packet-based file format and a replacement algorithm for the cache.

3. The Design of a Packet-Based Caching Proxy

Fig 1. illustrates the architecture of caching proxies. The deployment of well design caching proxies can save the backbone bandwidth significantly.

3.1. Cache File Format

Most video streaming applications use RTP as their transport protocol [11], so our caching focuses RTP packets. A basic RTP packet header is 12 bytes in size. Using the header, we can detect packet loss from the field of Sequence Number, and check the video format from Payload Type. We organize the cached packets in an index file scheme that is comprised of an Index File and a Packet File, with a sequence of packets composing a video clip. A caching proxy intercepts RTP packets and records them in the Packet File consecutively. The Index File consists of two columns: packet number and packet size. If a packet is received correctly, the packet number and size are written in columns, as shown in Fig 2. When a packet loss occurs, the packet size is recorded zero and will be repaired later. A caching proxy can access every packet easily according to the Index File, so VCR-functionalities can also be supported. The playing quality will be much better due to the QoS of LAN.

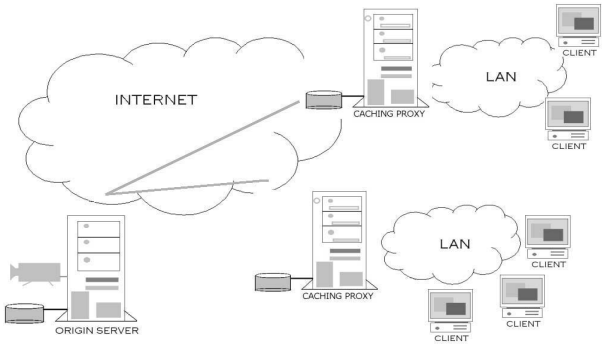


Fig 1. Internet video streaming with caching proxies

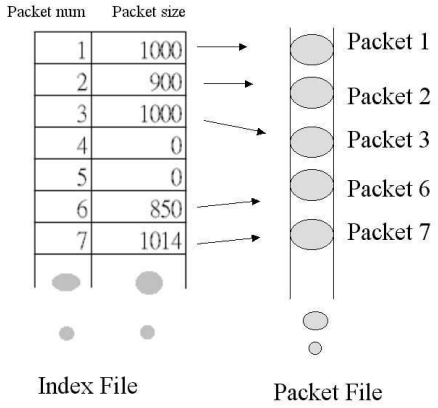


Fig 2. The format of Index File and Packet File

3.2. The Caching Scheme

We divide the caching scheme in four steps: step 1. Waiting and Streaming, step 2. Requesting, step 3. Caching and Forwarding, step 4. Repairing and Merging, as shown in Fig. 3. In step 1, the caching proxy wait for requests (consist of URL) from clients. If requests hit the cache, the proxy will stream packets from the cache to the client directly. If a request miss the cache or the request is a real time broadcast program, the proxy goes to step 2.

In step 2, Requesting, the caching proxy sends the request to upper proxies in the caching hierarchy. If none of these proxies has cached the video, the request arrives at the origin server. Then, the origin server starts to stream packets to lower caching proxies.

In step 3, Caching and Forwarding, the caching proxy receives packets from the upper proxy/server. If this is a broadcast program, the caching proxy can multicast these packets to all the clients in LAN. If these packets are from a video clip, the caching proxy must cache these

packets while forwarding to the clients in real time. The caching proxy also has to check the cache space and performs necessary replacements.

After the streaming session is complete, the caching proxy starts step 4, Repairing and Merging. Packets may get lost during streaming because of the RTP/UDP transmission. If we don't repair the lost packets, especially I-frame packets, the error will be propagated to lower caching proxies and clients. Therefore, we propose the repairing step to maintain loss-free cache. The caching proxy makes a TCP connection to the origin server and checks the Index File. From the "packet size" column of Index File, lost packets can be noticed and recovery requests can then be sent to the origin server. The origin server will find and resend the lost packets again. Packets won't get lost again because of reliable transmission of TCP. Thus the caching proxy can record the correct packet size and merge the lost packet with primary packets into a new "Packet File", as shown in Fig.4.

After completing the four steps above, proxies will have a loss-free packet cache of a video clip. Later requests of the same URL can be served directly from the caching proxy, and clients can receive low latency, good quality, and highly interactive video.

4. Implementation of the Caching Proxy

In this section we describe our implementation of the caching scheme. To demonstrate and experiment the caching scheme, we implement three components in C++ language: Origin Server, Caching Server, and Client. We also design a simplified streaming protocol based on the RTSP [12], and add our new repairing commands. We use H.263 bit stream as our video source, and the H.263 packets conform to the RTP payload format for H.263(+) video[13].

Fig. 5 shows the architecture of Origin Server. There are four modules: Server Daemon, Session Manager, Streaming Process, and Repairing Process. Server Daemon will make Session Manager to fork the corresponding process. Server Daemon and Repairing Process work on TCP connections.

The caching proxy architecture is presented in Fig. 5. Five modules are designed for a caching server. Direct Streaming Process streams RTP/UDP packets on LAN, while Caching & Forwarding Process cache packets from the Internet and forward to clients on LAN. Repairing Process will talk to another Repairing Process on Origin Server.

The Client is a graphics user interface program that accepts URLs from users and connects to a caching proxy. Then it will start program VIC, which is a Gnu Public Licensed video browser, to play video on the screen [14].

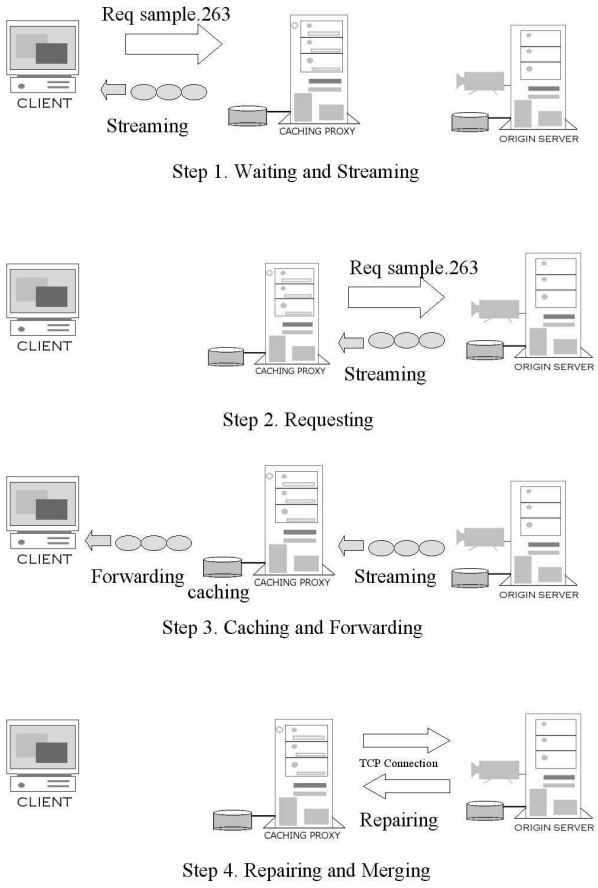


Fig. 3 The four Steps of video caching

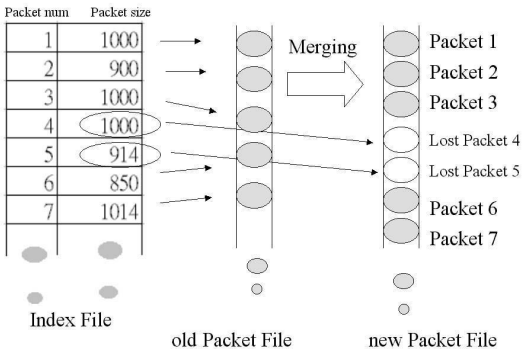
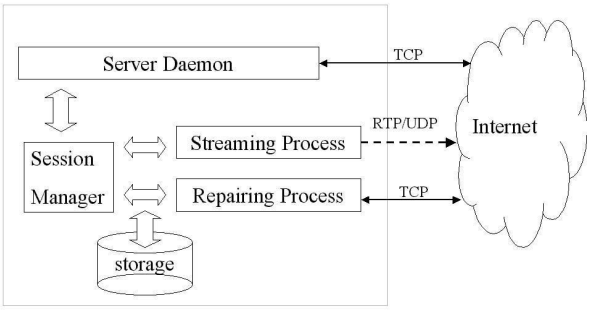
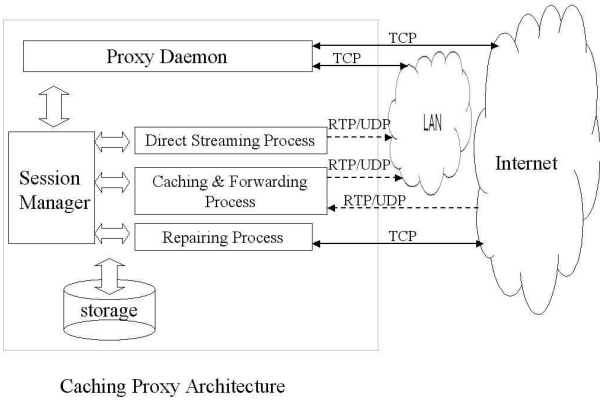


Fig. 4 Merging Packet File



Origin Server Architecture

Fig. 5 Origin Server Architecture



Caching Proxy Architecture

Fig. 6 Caching Proxy Architecture

5. The Cache Replacement Algorithm PLFU

When the cache space is full, the unused data need to be replaced with new data. The replacement algorithms of WWW and traditional video caching scheme focus on the whole object, but they are not suitable for longtime, continuous video streaming. If we keep all video clips in cache complete, the cache space is not enough for variety video clips in the Internet. Furthermore, clients play the initial part of a video clip more often than the whole. For example, the beginning part of a news program is accessed more frequently than the latter last. Therefore, the length of a video in cache is based on its popularity. The unpopular part of a video has higher priority to be replaced.

The replacement algorithm is designed to fit our packet-based caching proxy. We define a term, TP (Time Popularity), which represents the access frequency and access duration of the last part of a video.

In the past time t, $TP_{i,j}$ denotes the packet number played by the client j from the last packets of video i,. TP_i is the sum of all clients:

$$TPi = (\sum_{j=1}^{n} TPi, j)$$

Small TP_i means the last part of video i is not popular, and these packets should be removed.

We call our replacement algorithm PLFU (partial Least Frequently Used), as shown in Fig. 7. The PLFU algorithm is performed when the cache is running out of sapce. First, algorithm PLFU calculates TP for each video file in cache and choose the smallest one as the victim, which is to be partly removed. Second, algorithm PLFU removes a fixed number of packets, say k, from the victim. To prevent the "thrashing" phenomenon in cache, the victim can't be in use by any clients. If the victim is in use, we have to choose the next victim.

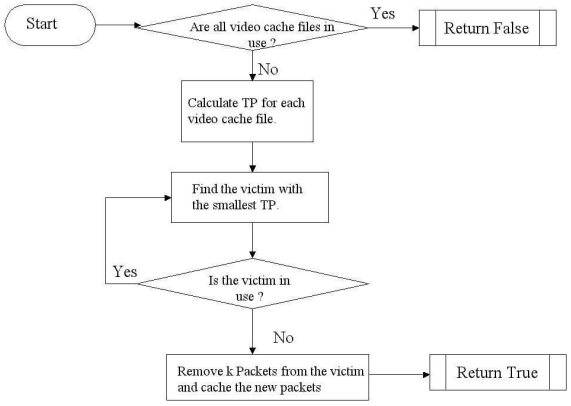


Fig. 7 PLFU Replacement Algorithm

6. PLFU Evaluation

The hit rate and the byte hit rate are usually used to evaluate the performance of a caching strategy. But it is not fit for our packet-based caching scheme. We suggest using the packet hit rate, which can reflect the new design. Different access patterns can affect simulation results. We simulate two kinds of access patterns here: random access pattern and Zipf distribution (θ =0.3) pattern [15].

Table 1. Simulation parameters

Simulation time	24 hours
Number of video	350 files
Video length	40 ~ 70 minutes
Video size	96MB ~ 168MB
Request rate	5 users per minute
Cache size(GB)	10, 15, 20, 25, 30, 35, 40
Last part x	2000*5 packets
Victim packet k	2000 packets
Packet size	1200 bytes

Simulation parameters are shown in table 1. Video length is 40 to 70 minutes, and we assume watching time of clients is uniformly distributed in 20%, 40%, 60% and 100% of video length. We compare PLFU with other well-known algorithms (LRU and LFU) through simulations [16]. All algorithms use packet-based caching, and the victim should not be in use to avoid thrashing.

The results show that our algorithm LPFU perform better than other algorithms both in packet hit rates and the number of replacement, especially in Random access, as shown in Fig. 8 to Fig. 11. This is because PLFU is smart to detect the situation where not all users watch the whole video, and the initial part can be preserved more in cache. LFU and LRU replace the whole cache file when the cache is full, and this will cause lower packet hit rate and more replacement packets. Because there is "hot" video effect, algorithms in Zipf distribution perform better than in Random Access.

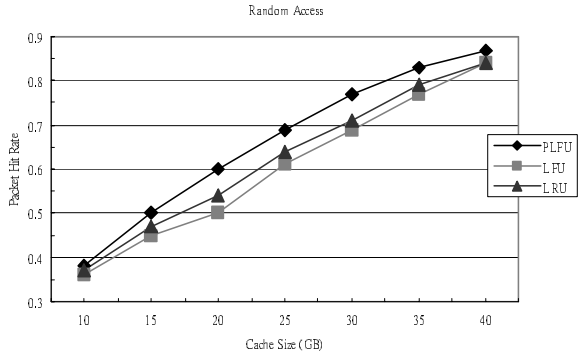


Fig. 7 Random Video Access: Cache Size vs. Packet Hit Rate

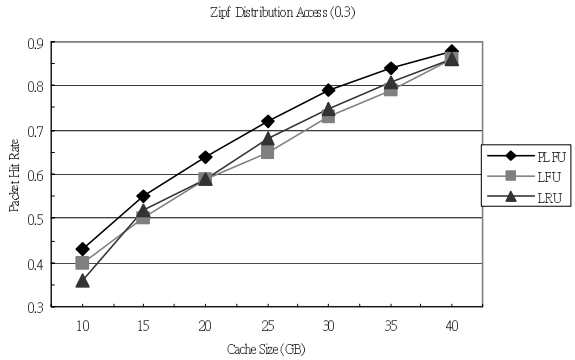


Fig. 9 Zipf Video Access: Cache Size vs. Packet Hit Rate

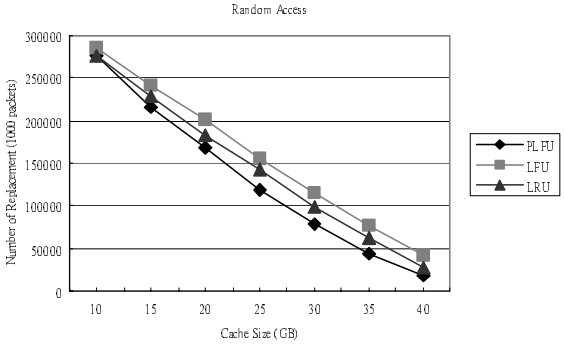


Fig. 10 Random Video Access (0.3): Cache Size vs Number of Replacement packets

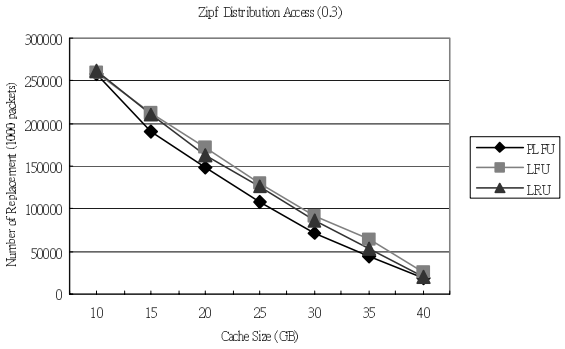


Fig. 11 Zipf Video Access (0.3): Cache Size vs. Number of Replacement packets

7. Conclusion

In this paper, we presented a packet-based caching scheme with a repairing step to minimize the error propagation. We also implemented Origin Server, Caching Proxy, Client, and designed a simplified streaming protocol for communication. In real experiments, we found that when compared with packet-based caching, file-based caching spent 150 times of CPU usage for the same video streaming. The reason is the extra cost payed for parsing and partitioning of H.263 video. We also proposed our PLFU replacement algorithm, and simulation results showed that it had better performance.

The use of caching proxies is the basis of Content Delivery Network (CDN). In the future, we will develop strategies for streaming packets to avoid causing network congestion.

8. References

[1] D. Eager, M. Vernon, and J. Zahorjan, "Optimal and efficient merging schedules for video-on-demand Servers," in *Proc. 7th ACM Multimedia Conf. (Multimedia '99)*, Orlando, Nov. 1999, pp. 199-202.

[2] K. Hua, Y.Cai, and S. Sheu, "Patching: A multicast technique for true video-on demand services," in *Proc. ACM Multimedia*, September 1998, pp. 191-200

[3] Zhi-Li Zhang, Yuewei Wang, D.H.C. Du, and Dongli Su, "Video staging: a proxy-server-based approach to end-to-end video delivery over wide-area networks Networking," *IEEE/ACM Trans. Networking*, Vol. 8, pp. 429-442, Aug. 2000.

[4] S. Sen, J Rexford, and D. Towsley, "Proxy Prefix Caching for Multimedia Streams," in *Proc. IEEE INFOCOM*, vol. 3, March 1999, pp. 1310-1319.

[5] J. Kangasharju, F. Hartanto, M. Reisslein, K.W. Ross, "Distributing layered encoded video through caches," in *Proc. IEEE INFOCOM*, vol. 3, 2001, pp. 1791-1800.

[6] R. Rejaie, Haobo Yu, M. Handley, and D. Estrin, "Multimedia proxy caching mechanism for quality adaptive streaming applications in the Internet," in *Proc. IEEE INFOCOM*, vol. 2, 2000, pp. 980-989.

[7] Eun-Ji Lim, Seong-Ho Park, Hyeon-Ok Hong, and Ki-Dong Chung, "A proxy caching scheme for continuous media streams on the Internet," in *Proc. of Int. Conf. Information Networking*, April 2001, pp. 720-725.

[8] Seong Ho Park, Eun-Ji Lim, and Ki Dong Chung, "Popularity-based partial caching for VOD systems using a proxy server," in *Proc. of Int. Conf. Parallel and Distributed Processing Symposium*, 2001, pp. 1164-1168.

[9] V. Hilt, M. Mauve, and W. Effelsbert, "A light-weight repair protocol for the loss-free recording of MBone sessions," in *Proc. of Int. Conf. Distributed Computing Systems Workshop*, April 2001, pp. 63-68.

[10] M. Zink, A. Jonas, C. Griwodz, and R. Steinmetz, "LC-RTP (loss collection RTP): reliability for video caching in the Internet," in *Proc. of Int. Conf. Information Networking*, April 2000, pp. 281-286.

[11] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *RFC 1889, IETF Network Working Group*, January 1996

[12] H. Schulzrinne, A. Rao, and R. Lanphier, "Real time streaming protocol (RTSP)," *RFC 2326, IETF Network Working Group*, Feb. 1998.

[13] Bormann, L. Cline, G. Deisher, T. Gardos, C. Maciocco, D. Newell, J. Ott, G. Sullivan, S. Wenger, and C. Zhu, "RTP Payload Format for the 1998 Version of ITU-T Rec. H.263 Video (H.263+)," *RFC 2429, IETF Network Working Group*, October 1998

[14] Video Conference Tool, Vic, http://www-mice.cs.ucl.ac.uk/multimedia/software/

[15] L.Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," in *Proc. IEEE INFOCOM*, vol. 1, 1999, pp. 126-134.

[16] L. Rizzo, and L. Vicisano, "Replacement policies for a proxy Cache," *IEEE/ACM Trans. Networking*, Vol. 8, pp. 158-170, April. 2000.