Peer-to-peer Streaming of Stored Media: The Indirect Approach

Tadeusz Piotrowski, Suman Banerjee
University of Wisconsin-Madison, WI 53706, USA
{tadeusz,suman}@cs.wisc.edu

Sudeept Bhatnagar, Samrat Ganguly, Rauf Izmailov
NEC Laboratories, Princeton, NJ 08540, USA
{sudeept,samrat,rauf}@nec-labs.com

Categories and Subject Descriptors: C.2.4 [Distributed Systems]: Distributed applications

General Terms: Algorithms, Design, Measurement, Performance.

Keywords: Media-streaming, Peer-to-peer, Overlays.

1. INTRODUCTION

We present Π-Stream, a system which enables fast real-time media-streaming that allows clients to take advantage of multiple servers, when available, as well as allows peer-to-peer style collaborations between such clients for improved performance This is in contrast to current commercially available media-streaming players such as RealPlayer or Quicktime are limited to downloading from a single server. Our work is motivated by the continued increase in available network *access* capacity in recent years, that has led to proliferation of rich multimedia content over the Internet. In many cases, this has moved the Internet path bottlenecks from access links to intra-AS and peering links [1]. Π-Streamproposes to circumvent these bottlenecks by exploiting opportunities for parallel downloads from multiple servers and client peers — an approach that has gained popularity today for large file transfers, e.g., in BitTorrent (see www.bittorrent.org). While the principle driving BitTorrent and Π-Streamare fairly similar, the algorithmic solution and implementation requirements of these two systems are fundamentally different due to differences in objectives of the respective applications. A file transfer application is primarily latency-insensitive. Hence, an efficient peer-to-peer file transfer mechanism needs to answer the question of *what* 'block' of data needs to be downloaded from *which* peer. In contrast, a media streaming application is latency-sensitive, and needs to answer the question of *when* to download an individual data block in addition to the *what* and *which* questions.

A key design objective in Π-Stream has been the following requirement: a user of Π-Stream should be able to use any existing (commercially deployed and widely available) media-streaming client software and get the above benefits. In doing so, the user should not need to make any changes to the client software itself.

Π-Stream achieves this objective by incorporating a *Local Proxy Stream Server* or LPSS between a set of servers and peers and the client (we will henceforth refer to downloads from static servers as LPSS-Standalone and combined downloads from static servers and peers as LPSS-Torrent). The LPSS consists of two independent stages located at the client (i) an LPSS downloader that utilizes a simple protocol such as HTTP to timely schedule the download of media content from a set of servers and other clients; and (ii) an LPSS streamer that behaves like a streaming server and continually streams downloaded content to the client's local media player (Figure 1). Through this process of indirect streaming, the LPSS is able to ensure protocol independence between the media providers and the client player as well as graceful recovery from server failure. Indirect streaming also allows the LPSS to use disk storage on the client to limit wasted bandwidth, thus making its streaming speed limited only by the access bandwidth.

The main advantage of Π-Stream over other proposed distributed streaming system solutions such as PALS [4] and PROMISE [2] is that indirect streaming allows Π-Stream to communicate with different servers using different protocols such as (HTTP, FTP) without the media client being aware of them. Also, the use of TCP to retrieve blocks from servers and clients requires no additional congestion control mechanism. Furthermore, Π-Stream's ability to interoperate with existing web servers and media-streaming clients makes Π-Stream easily deployable in wide-area settings.

2. ARCHITECTURE AND ALGORITHMS

There are three different components in the Π-Stream architecture. (i) *The Client*, which includes the LPSS and the media player that are co-located in the same machine as independent processes, (ii) *The Server*, this can be any byte-stream server, e.g, a web server or a FTP server, which stores the entire media content and serves the clients in small units of blocks on demand, (iii) *The Tracker*, which is analogous to the tracker used in BitTorrent, it provides information to each client about the location and the download status of other clients in the system that are requesting playback of the same media content.

A media player initiates the playback by sending an appropriate request to its local LPSS. The LPSS in turn contacts the appropriate tracker. The tracker responds with location information of a few (web) servers that store the relevent content of the file as well as the location of other peer LPSS nodes involved in playback of the same content. The LPSS now has two jobs. First, it needs to appropriately identify a relevant subset of servers and other peer LPSS from which it should schedule a download. Second, given this set of chosen servers and peer LPSS, it must plan a download schedule of different blocks and adapt the schedule to changes in net-

SIGMetrics/Performance'06, June 26–30, 2006, Saint Malo, France.
ACM 1-59593-320-4/06/0006.

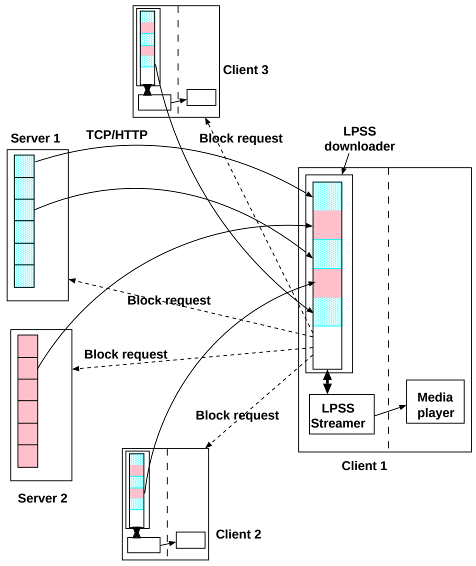


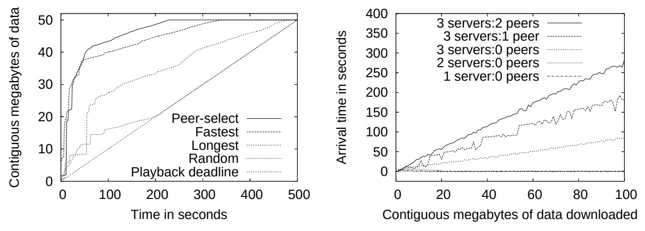
Figure 1: The Π-Stream architecture.

work conditions. These two task are described in greater detail below.

Block Scheduling Algorithm: This algorithm adaptively identifies blocks that need to be downloaded from different servers and peers to meet the playback deadline. The goal of this algorithm is to minimize the probability of a block not being available at its playback time. This algorithm is invoked in two cases: 1) When the rate monitor finds that a block being downloaded might miss its playback deadline, 2) when a server finishes its assigned block and the LPSS has to assign a new block to that server.

In solving the block transfer scheduling problem, we employ the following approach: 1) Get a given block at the earliest possible time subject to all previous blocks arriving at the earliest and its own playback deadline requirements being met, 2) Try to get any block in its entirety in a single request from one server, 3) Ask for sub-blocks only if the block's deadline is not likely to be met if it is downloaded as a whole. To satisfy the above constrains we first employ an Earliest Finish Assignment for each block. Next, if a block is going to miss its playback deadline, we use connection swapping to reassign a block with an earlier deadline to a fast server and a block with a later deadline to a slower server. Finally, if a block is still going to miss its deadline we proceed to split the block. The idea here is that the block can now be downloaded from two servers in parallel, reducing the total download time and resulting in the playback deadline being met.

Peer Selection Algorithm: This algorithm maximizes the amount of useful data the LPSS peers can provide. We use three parameters to decide on the choice of peers and servers to download from. The first parameter is the sustained transfer rate between the peer (or server) and the requesting LPSS. The next two parameters only apply to peer selection using LPSS-Torrent. These are (i) the difference in playback time between the requesting LPSS and the peer LPSS - since the media download proceeds linearly the farther ahead the peer LPSS is in the download process, the greater is the amount of additional data it can contribute;



(a) Peer selection. (b) Varying peers and servers.

Figure 2: Performance evaluation on PlanetLab.

and (ii) the duration of time the other LPSS is expected to stay in the system - a peer that is expected to disappear quickly is expected to be less useful to the requesting LPSS. Therefore, among multiple alternate choices in a set of peers, we always choose a peer that can provide the highest amount of data to the requesting peer.

3. EXPERIMENTS AND SUMMARY

Our evaluation of the Π-Stream architecture is done through extensive testing of a prototype implementation on PlanetLab. The following is a summary of our main results:

- Achieved a sustained video playback rate of more than 2.4 Mbps using parallel downloads from multiple servers. Established the Block Scheduling Algorithm's ability to retrieve all file blocks before their playback deadline.
- Demonstrated the efficacy of the proposed Peer Selection Algorithm (see Figure 2(a)) in comparison to (i) selecting a random peer (random), (ii) selecting the peer with longest remaining time in the system (longest), and (iii) selecting the peer with the fastest download rate (fastest).
- Illustrated Π-Stream's resilience to varying network conditions by sustaining a set playback rate as different servers failed and joined.
- Quantified performance improvements at clients as the number of servers and peers in the system increased. In particular, we demonstrated significant performance benefits of peer-to-peer collaborations, i.e., in the LPSS-Torrent mode. (Figure 2(b) illustrates how increasing the number of servers and simultaneous peers increases the arrival of blocks prior to the desired deadline.)

A detailed analysis of our results, including plots can be found [3].

4. REFERENCES

[1] S. S. A. Akella and A. Shaikh. An empirical evaluation of wide-area Internet bottlenecks. In *Internet Measurement Conference*, 2003.

[2] M. Hafeeda, B. Habib, A. Botev, D. Xu, and B. Bhargava. Promise: Peer-to-peer media streaming using collectcast. In *Proceedings of ACM Multimedia*, 2003.

[3] T. Piotrowski, S. Banerjee, S. Bhatnagar, S. Ganguly, and R. Izmailov. Peer-to-peer streaming of stored media: The indirect approach. Tech. Report TR 1561, Computer Sciences Department, UW-Madison, Apr. 2006.

[4] R. Rejaie and A. Ortega. Pals: Peer-to-peer adaptive layered streaming. In *NOSSDAV*, 2003.