GNUSTREAM: A P2P MEDIA STREAMING SYSTEM PROTOTYPE

Xuxian Jiang Yu Dong Dongyan Xu Bharat Bhargava

Department of Computer Sciences
Purdue University, West Lafayette, IN 47907
E-mail: {jiangx,dong,dxu,bb}@cs.purdue.edu

ABSTRACT

We present the design and prototype of *GnuStream*, a peer-to-peer (P2P) and receiver-driven media streaming system. GnuStream is built on top of *Gnutella*, and it integrates dynamic peer location and streaming capacity *aggregation*. Each GnuStream streaming session is controlled by the receiver peer and involves a *dynamic* set of peer senders instead of one fixed sender. The receiver aggregates streaming bandwidth from the multiple senders, achieving load distribution and fast reaction to sender capacity and on/off-line status changes. The effectiveness of GnuStream is demonstrated by our experiments with its prototype, which will serve as the basis for real-world development and evaluation of resilient P2P media streaming services.

1. INTRODUCTION

The concept of P2P has recently gained popularity thanks to the wide deployment of P2P file sharing applications over the Internet. We are interested in the application of P2P to real-time media streaming, which is a more challenging problem than ordinary file-sharing. In file-sharing systems, a client first downloads the *entire* file and then uses it, termed as the "open-after-downloading" mode; while in media streaming systems, a client consumes the content of a media file *while* the file is being downloaded, termed as the "play-while-downloading" mode [1].

The large volume of media data along with their stringent timing constraint poses challenges for P2P media streaming, which is even exacerbated by the dynamic nature of P2P networks: a peer can get online or offline at any time. Furthermore, a single peer sender may not be able to contribute full streaming bandwidth to a peer receiver, due to its limited capacity and/or its own communication needs. As a result, the traditional single-sender paradigm is no longer effective in P2P streaming. Instead, a P2P streaming session has to involve a *set* of peer senders, each contributing a *portion* of the streaming bandwidth. In addition, different from the client-server paradigm [2], the sender set members may *change* dynamically, due to their unpredictable online/offline status changes.

Recently, P2P streaming has been studied from other angles. Narada [3] and PeerCast [4] are two proposed architectures for synchronous broadcast of live media to multiple peers. They focus on dynamic construction of a multicast tree which connects peers requesting the live media. However, they do not consider the limited contribution of bandwidth from individual peers. CoopNet [5] builds multiple distribution trees spanning a source and all peer receivers, each tree transmitting a separate description of the media signal. However, each peer receiver consults the *source* node for its upstream peer senders. ZIGZAG [6] organizes peer receivers into a hierarchy of bounded-degree clusters and builds a multicast tree based on the hierarchy. However, it assumes that each peer only receives from one upstream peer sender.

In this paper, we present the design and implementation of GnuStream, a *receiver-driven* P2P media streaming system. Built on top of Gnutella, GnuStream takes into consideration the underlying P2P network dynamics and heterogeneity. It features multi-sender bandwidth aggregation, adaptive buffer control, peer failure or degradation detection and streaming quality maintenance. To the best of our knowledge, GnuStream is the first prototype with all the features above. The rest of paper is organized as follows: Section 2 gives an overview of GnuStream. Section 3 presents the implementation of GnuStream, followed by experimental results in Section 4. Finally, Section 5 concludes this paper.

2. SYSTEM OVERVIEW

2.1. System Environment

Figure 1 shows the GnuStream system environment: The underlying P2P network is Gnutella. Suppose peer P1 is looking for media file *X*. After calling the Gnutella lookup service, P1 locates four candidate senders P2-P5.

If the aggregated bandwidth of P2-P4 (contributing different amounts of bandwidth as indicated by the edge thickness) is sufficient for streaming *X* in full quality, P5 will become a standby sender, to be called upon to take over the load of degrading/disconnected peer senders during the streaming session.

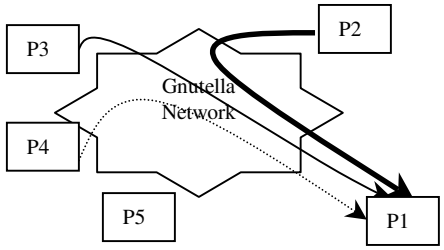

Figure 1: GnuStream system environment

2.2. System Features

GnuStream has the following salient features, which are either missing or not yet implemented in other P2P media streaming systems:

(1) **Integration with P2P lookup substrate**: GnuStream leverages Gnutella as its lookup substrate, making it readily deployable in the current Gnutella P2P network environment.

(2) **Multi-sender aggregation**: Instead of relying on one single sender, GnuStream distributes streaming load among multiple peer senders. The following two load allocation policies have been implemented in GnuStream:

- *Even allocation:* to distribute streaming load evenly among peer senders. This policy is suitable for a well-provisioned and homogeneous environment such as company Intranet.
- *Proportional allocation*: to distribute streaming load in proportion to the current capability of peer senders. This policy is more flexible and adaptive than even allocation, making it suitable for a dynamic and heterogeneous environment such as the wide-area Internet.

(3) **Receiver data collection**: The receiver coordinates the arrival of different media data segments, and re-constructs media data in their original and continuous order before feeding them to the media player.

(4) **Detection of peer status change**: GnuStream uses periodic probing and soft states to detect any changes in the status of peer senders, including peer disconnection, failure, and bandwidth degradation.

(5) **Recovery from failure or degradation**: If a current peer sender is detected as suffering from failure or bandwidth degradation, GnuStream will migrate all or part of its streaming load to another peer sender or a standby peer sender (such as P5 in Figure 1). Therefore, the set of active peer senders in one streaming session will change dynamically during the session.

(6) **Buffer control**: To accommodate the dynamic set of peer senders and the end-to-end network congestion, GnuStream implements a suite of buffer control mechanisms which involves more concurrency and scheduling complexity than the traditional buffer control mechanisms in client-server streaming.

3. DESIGN AND IMPLEMENTATION

In this section, we present a "zoom-in" description of GnuStream's design and implementation, highlighting its layered architecture and buffer control mechanisms.

3.1. Three-layer Architecture

GnuStream consists of three layers: Network Abstraction Layer, Streaming Control Layer and Media Playback Layer. The architecture is shown in Figure 2.

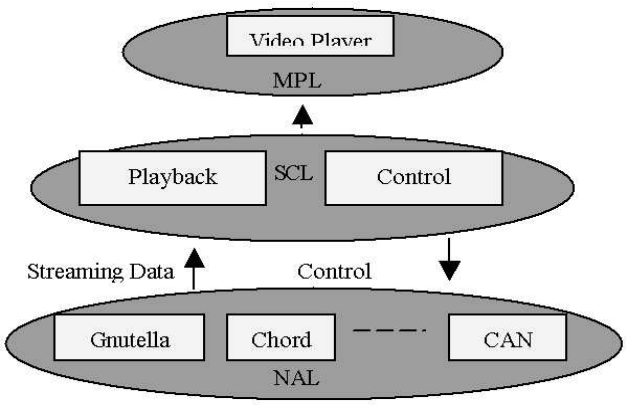

Figure 2: GnuStream architecture

(1) **Network Abstraction Layer (NAL)**: NAL masks underlying P2P network peculiarities and provides a generic and uniform interface of P2P lookup substrate. NAL performs efficient peer and content discovery for P2P media streaming. Furthermore, NAL makes GnuStream not only deployable in Gnutella, but also portable to other P2P networks.

(2) **Streaming Control Layer (SCL)**: SCL deals with the dynamics and heterogeneity of P2P networks. At the core of GnuStream, SCL performs the key functions of bandwidth aggregation, data collection, and status change detection and recovery. The dynamic peer

sender set is also maintained and adjusted by SCL. The goal is to maintain the maximum streaming quality despite the dynamics in the underlying P2P network. Buffer management (to be described in the next section) is the critical technique to achieve this goal.

(3) **Media Playback Layer (MPL)**: To be adaptable to the aggregated streaming bandwidth, MPL performs media quality adaptation based on the media data collected by SCL. The technique of double buffering is used between media decoder and player for efficiency and low switching overhead. MPL also enables the "plug-n-play" of different media players with minimum modification.

3.2. GnuStream Implementation

We have implemented GnuStream using Microsoft Visual C++ 6.0 and leveraging the open source Gnutella client, namely *Gnucleus*.

The most challenging aspect of our implementation is buffer management. Traditional buffer control in the client-server paradigm cannot be applied directly, due to the P2P network dynamics. Figure 3 illustrates the buffer hierarchy in GnuStream to deal with P2P network dynamics, in which the *control buffer* is the core of buffer control mechanisms. Double *display buffers* and *decode buffer* reside in MPL, which speed up display and reduce the synchronization overhead between MPL and SCL. *Data collection buffer* at SCL is used to eliminate data overflow and underflow and smooth the data arrival jitter in the presence of multiple peer senders.

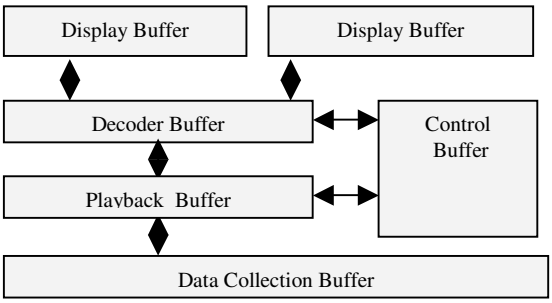

Figure 3: Buffer management hierarchies

Inside the *control buffer*, to coordinate the multiple peer senders, a delicate buffer model is implemented to enforce the real-time property of media streaming. Under this model, we manipulate the control buffer using different control pointers. The *offset* pointer indicates the amount of data already consumed. *End-of-data* indicates the trunk of continuous data available in the buffer. *End-of-buffer* indicates the end or maximum size of buffer. SCL adjusts the data feeding rate to the decoder and thus the display frame rate is tunable according to current aggregated streaming bandwidth from the multiple peer senders. With this model, SCL is also able to monitor the streaming progress and thus compute system parameters such as the next frame needed and expected bit-rate from each peer sender.

4. EXPERIMENTS

We create a local experimental testbed using multiple desktop PCs with XEON CPU 2.00 GHz, each equipped with 1.00G RAM and 3COM 3C920 100Mbps Integrated Fast Ethernet Controller. Some of them are configured as peer senders while the others are configured as the requesting receivers. We limit the maximum streaming bandwidth from each sender to 60KB/s. Figure 4 is a snapshot showing the status of peers participating in Gnutella network, as well as the video image from the receiver of a P2P streaming session.

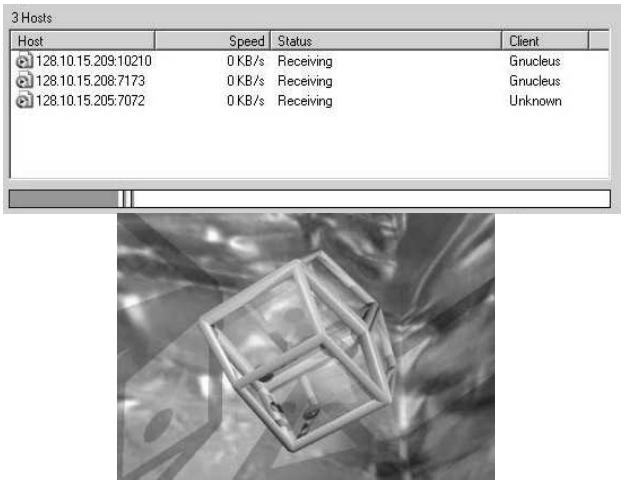

Figure 4: A snapshot of GnuStream in operation

(1) **P2P streaming session setup** Before a P2P streaming session starts, the receiver uses the following equation to compute the aggregated streaming bandwidth and then determines how many peer senders are needed.

$$Arrival\ Media\ Bandwidth = \frac{Frame\ Size * Bits\ per\ Pixel * YUV\ Sampling * Playback\ Rate}{Compression\ Rate * Bits\ per\ Byte}$$

The test media clip in our experiments has the following parameters: image size 352*240, frame rate 30frames/second, YUV sampling 4:2:0, 8 bits per pixel,

and compression rate 26:1. The resultant streaming bandwidth is approximately 143.0KBps. Therefore, the receiver contacts three peer senders (discovered by Gnutella lookup service), each willing to provide a streaming bandwidth of 60KBps. The receiver then initiates the streaming session and receives full-quality streaming quality.

(2) **Buffering for continuous media playback** To achieve continuous and jitter-free media playback, the receiver performs buffering both *before* and *during* the media playback. The initial buffering delay can be determined by the following equation:

$$\frac{Initial \quad Delay}{Number \quad Of \quad Frames} = \begin{cases} 0 & if \quad AFR \geq PFR \\ \frac{1}{AFR} - \frac{1}{PFR} & otherwise \end{cases}$$

where

AFR : *Arrival Frame Rate*

PFR : *Playback Frame Rate* (*typically* 30 *fps*)

Our test video clip has 1680 frames. Suppose the AFR is 25 fps, we can compute the initial buffering delay to be 11.2 seconds. Note that during the initial buffering, all peer senders transmit media data at the same rate as their contributed streaming bandwidth. In addition, notice that the aggregated streaming bandwidth (3*60KBps) is greater than the playback rate (143.0KBps): each peer sender will exploit the residual bandwidth to pre-transmit media data which will be buffered by the receiver during the media playback.

(3) **Peer failure detection and recovery** During the streaming session, we intentionally turn off the Gnutella services on one of the peer senders, in order to test the responsiveness of GnuStream's peer failure recovery function. In the 27th second of the streaming session, Peer 3 is turned off. It takes GnuStream approximately 1.0 second to detect and recovery from the failure, as shown in Figure 5. Notice that Peer 3 (represented by the magenta curve) is replaced by Peer 4 (cyan curve), which has been a standby sender during the first 26 seconds. The detection and recovery latency is a result of the tradeoff between system reactivity and peer probing overhead. We note that there is *no* interruption to media playback even during the 1.0 second period of sender set adjustment, thanks to the buffer control mechanisms of GnuStream.

5. CONCLUSIONS

P2P media streaming is expected to be widely deployed in P2P networks such as Gnutella. In this paper, we present the design and implementation of GnuStream, a P2P and receiver-driven media streaming system which is readily deployable in Gnutella network. GnuStream is aware of the dynamics and heterogeneity of P2P networks, and leverages the aggregated streaming capacity of individual peer senders to achieve full streaming quality. GnuStream also performs self-monitoring and adjustment in the presence of peer failure and bandwidth degradation. Our experiments have demonstrated the effectiveness of GnuStream buffer control mechanisms in maintaining both continuity and quality of P2P streaming sessions. Finally, GnuStream is an open source software system, which can be used as a basis for the implementation and evaluation of more advanced and complex features of P2P media streaming.

We have released GnuStream at: http://www.cs.purdue.edu/homes/jiangx/GnuStream/

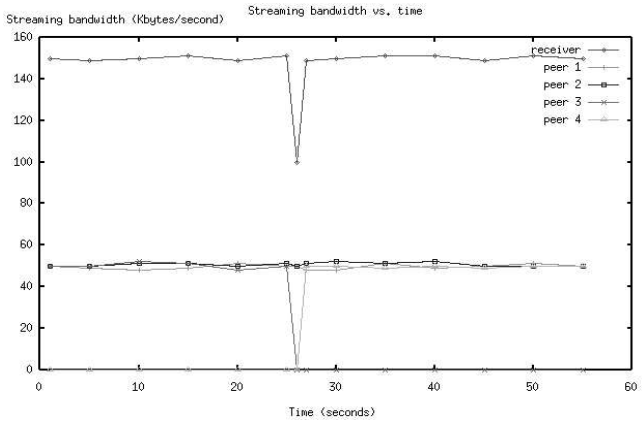

Figure 5: Peer failure detection (in 27th second) and recovery (Peer 3 replaced by Peer 4)

6. REFERENCES

[1] D. Xu, M. Hefeeda, S. Hambrusch and B. Bhargava, "On Peer-to-Peer Media Streaming", *IEEE ICDCS 2002*, July 2002.

[2] T. Nguyen and A. Zakhor, "Distributed Video Streaming Over Internet", *SPIE/ACM MMCN 2002, Jan*. 2002.

[3] Y. Chu, S. Rao and H. Zhang, "A Case for End System Multicast", *ACM SIGMETRICS*, June 2000.

[4] 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.

[5] V. N. Padmanabhan, H. J. Wang, P.A. Chou and K. Sripanijkulchai, "Distributing Streaming Media Content Using Cooperative Networking", *ACM NOSSDAV*, May 2002.

[6] D. A. Tran, K. A. Hua and T. T Do "ZIGZAG: An Efficient Peer-to-peer Scheme for Media Streaming", *IEEE INFOCOM2003*, April 2003.