UBC CPSC 416 Distributed Systems: Project Proposal

Alcuaz, Ito Franchilo Mikael[1] — Aziz, Shariq[2] — Ko, Mimi[3] — Musa, Abrar[4]

Abstract

Media streaming refers to the constant delivery of time-ordered multimedia content from a server to a client. It is an attractive option as it provides instant access for the end user and is quite flexible – a user doesn't have to download the entire file beforehand as was traditionally done in the past, and can interact with the data as it arrives. Naturally, such a system demands a high bandwidth connection and so works more effectively for users on faster internet networks. This proposal serves as an outline for a possible solution to this key shortcoming by de-centralizing the server-client model through the use of peer-to-peer (P2P) stream forwarding.

[1] *14633127 (y9u8), ialcuaz@alumni.ubc.ca*
[2] *14694146 (i2u9a), shariqazz15@gmail.com*
[3] *66666666 (o3d7), mimi@dbzmail.com*
[4] *66666666 (i1u9a), abrar.musa.89@gmail.com*

Contents

1. Introduction

A P2P implementation of a media streaming system alleviates some of the bandwidth requirements due to the fact that content is distributed over several streams rather than over one singular server. Usually the content is multicasted from a server to some client which has requested the stream; YouTube makes use of this notion by implementing multiple CDN's which distribute the content by demand, over a single connection which demands a high bandwidth for maximum playability. Therefore with P2P streaming (similarl to how P2P BitTorrent clients work) the system becomes more scalable – as the number of clients increase, so do the number of potential peers which can now act as a source for part of the streamed media. The server and clients together form a network of media streams.

Such a system, however, must preserve an invariant trait: clients must be able to connect and disconnect at will – it must be able to discover the parent server through a given URL, and as well, the system must be flexible enough to handle and detect a disconnected client. There must also exist logic to decide which clients copy parts of the stream, and application level functionality to allow a client to reject P2P connections (similar to rejecting seeding in BitTorrent applications).

A popular P2P implementation is PeerCast (now defunct) and FreeCast, both of which will be examined and used as a model for this group's implementation using the Go programming language.

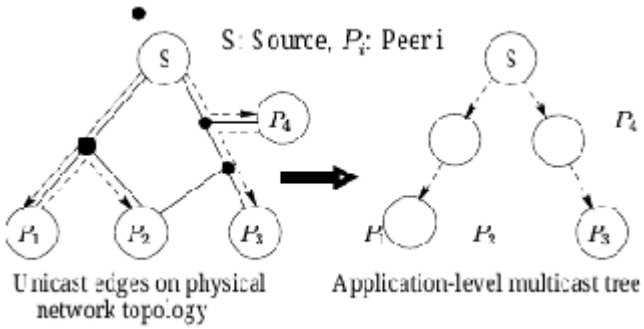

Figure 1. A multicast tree. [1]

2. Implementation Strategy

2.1 Software engineering practices

To follow good industry practices, this group will probably follow test driven development (TDD) methods, and SCRUM based procedures so that the project and features are clearly outlined and implemented with minimal risk.

2.2 Construction of a single, centralized server, and requesting clients

This is the base case of the problem statement and should be implemented first and foremost. A server with a dummy file will be set up and a client will connect and stream the file to completion through a connection. To closely mimic the standard video transferring, a TCP connection will be first set up between the two on the initial handshake phase, and then use UDP packets to stream the data. It is critical that the packets be time-ordered. The server should be able to handle an arbitrary number of clients, and detect disconnections and failures.

2.3 Stream forwarding policies

Clients must be able to replicate the stream or parts of it from the server. An algorithm to decide where the server forwards its stream will also be implemented; the amount of forwarding must scale proportional to the number of connected clients. As above, failures and disconnects must be detected, such as the case of a replicated stream that is lost.

2.4 P2P streaming, edge case failures, and application level functionality

Clients will then be able to stream the data through other clients; there must exist an algorithm to decide when to do so, and the client which is streaming data to another client must communicate with the server to avoid data redundancy. This is the point in the project where the discussion and edge cases discussed in [1] and [2] will come into fruition, and as such, will likely take significantly more time than the other segments. It would also be a nice plus to implement a feature which allows a client to prevent itself from acting as a source.

2.5 Project deployment and extras

In this section the GUI will be implemented, and the project will be hosted on a free website, possibly Heroku. It is hoped that any user can then upload multimedia content to some centralized server (maybe their machines can also act as servers ?), of which other users can then stream using the algorithms implemented above.

3. Project Timeline

To allow as much time as possible for polishing, debugging, and unforeseeable events, the meat of sections 2.2 - 2.4 should be completed on, or 1-2 weeks after Mar. 18, which corresponds to the project status meeting date. Step 2.2 most definitely needs to be finished as soon as possible within the next few weeks and shouldn't take as long as the subsequent steps.

4. SWOT Analysis

4.1 Strengths

There is a wealth of resources available on P2P streaming and established solutions which would serve to simplify the design and implementation process. The P2P system self scales as more nodes join, which is at the heart of distributed systems.

4.2 Weaknesses

The system will be difficult to test and complicated to build due to the replication procedures, P2P links, etc.

4.3 Opportunities

The Go programming language was created to specifically build distributed systems. As was seen in the class assignments, standard networking procedures are fairly straightforward to implement using this language.

4.4 Threats

The Go programming language is fairly new which makes solutions to specific problems harder to come by than other well established languages, and the members of this group have not had experience with it prior to taking this course.

5. Conclusion

A solution to the drawback of the centralized client-server model in media streaming is found through the use of P2P networking. This is achieved by sharing the bandwidth requirements across several disjoint nodes which serve to forward the data stream among themselves, not over one singular connection.

References

[1] H Deshpande. *Streaming Live Media over Peers*. 2002.

[2] Z Shen. Peer-to-peer media streaming: Insights and new developments, 2011.