Live Streaming on P2P Networks: Midterm Report

Atif Nazir

Topics in Internet Research (Spring ’06)
Lahore University of Management Sciences

Introduction

Live streaming\(^1\) of multimedia has become a popular choice for users on the internet, especially with the emergence of services like IPTV. Researchers have only recently started concentrating on addressing challenges (such as scalability and playback continuity) posed by peer-to-peer live media streaming.

All solutions proposed for live streaming in a P2P environment utilize either mesh-based P2P overlays or tree-based P2P overlays\(^2\). We consider the case of AnySee\(^2\) and observe that the architecture can be modified slightly to improve its resilience to node failures – this shall be called Modified AnySee. We observe, further, that while literature on AnySee and Coolstreaming\(^1\) mentions a “required” buffer of 40s and 120s, respectively, for “satisfactory” playback continuity (of video streams), no work, to the best of our knowledge, has attempted a quantitative study of the reasons for the “required” buffer sizes and the impact of changes in buffer sizes on the architectures’ performance under stable and dynamic conditions.

This report aims to address the first of these issues: a modification to the existing AnySee architecture to improve its performance under dynamic conditions (where peers may leave the network unannounced). For the sake of completeness, we provide brief overviews of the Coolstreaming and AnySee approaches followed by Modified AnySee. We conclude this report with a plan for addressing the second issue, namely the buffer analysis of Coolstreaming, AnySee and Modified AnySee.

---

1 A media stream can be on demand or live. On demand streams are stored on a server for a long period of time, and are available to be transmitted at a user's request. Live streams are only available at one particular time, as in a video stream of a live sporting event. [Wikipedia]
2 We shall restrict the scope of this project to two such approaches: AnySee\(^1\) and Coolstreaming\(^2\).
Coolstreaming: A Mesh-based Approach

Coolstreaming relies on establishing a mesh of peers with one or more serving nodes (seeds) that supply data to the rest of the data. The mesh overlay is constructed such that each peer has a path to every other peer in the network - each peer may retrieve or transmit stream data to other peers (that are directly adjacent to them in the overlay), however, seeds may only transmit stream data to peers directly adjacent to them in the overlay network. The following figure shows an example mesh overlay for Coolstreaming.

In this example, node A is the seed while the rest of the nodes are peers.

Each media event, in Coolstreaming, results in the formation of a separate overlay where the seed(s) are responsible for initiating the transmission of the stream to other nodes in the overlay. A node X may join a Coolstreaming overlay by contacting the seeder (node A in our case) which provides node X with information about one other node (referred to as the seed’s deputy) participating in the stream. Node X contacts node A’s deputy, retrieves a partial list of participating nodes and connects to a number, say five, of them which then become node X’s neighbors.

Coolstreaming breaks its media stream into segments, which the seed transmits sequentially to a randomly chosen set of its neighbors. All nodes advertise the list of segments currently stored on them to their neighbors, who then pull or ask for the relevant, available, segments from the neighbor (who advertised its list), and try to maintain a steady playback rate for the user(s).

---

It is to be noted that the neighbors are chosen regardless of the quality of service they can provide (initially). Nodes are free, however, to explore establishment of connections with other nodes in the overlay once their existing neighbors are judged to be “leeches” or neighbors with “bad” upload-download ratio – this cannot be determined before exchanging stream segments and hence connections with neighbors are always formed regardless of the type of quality of service that can be provided by them.
Messages concerning appearance of new nodes into the overlay and departure of old nodes from the overlay are broadcasted across the overlay network through an improved version of the gossip protocol (Scalable Gossip Membership Protocol or SCAM)—this is done until all nodes in the network have received information about the joining/leaving of the nodes. In case of node failure, the failed node’s neighbors are responsible for gossiping a “leave” message to other peers in the overlay— a neighbor is expected to have failed once it stops issuing “keep-alive” messages to its neighbors.

The literature on Coolstreaming fails to provide exact reasons for its suggested buffer size of 120 seconds, but a large buffer can be expected to be required due to the lack of stability of data rates or latency in reception of stream segments. While Coolstreaming tries to minimize problems in playback continuity by scheduling requests for segments with lower availability first (i.e., the stream segments may be present only on a small number of neighbors with low data rates), it still cannot guarantee their reception in a timely manner—it is thus that Coolstreaming’s playback continuity suffers drastically under dynamic conditions (where peers may leave unannounced).
AnySee: A Tree-based Approach

In AnySee, each seed forms its own separate tree-based overlay for each media event it must stream to peers in the overlay. There may be more than one seeds in the AnySee network, where peers participating in one overlay may additionally participate in overlays of other seeds – i.e., a peer maintains a set of “active streams” or connections in overlays through which it is actively receiving the media stream\(^4\), and a set of “backup streams” or connections in overlays other than those with “active streams” (these connections, although established, are inactive). Furthermore, all trees in the AnySee network are formed on the basis of delays\(^5\) – i.e., an upper limit on the delay, say 10ms, is defined and connections are made in the overlay(s) such that every participating node has a path of at most 10ms delay to the seed(s), and the path established provides the minimum source-to-end delay for that participating node. The following figure provides an example AnySee network:

\(^4\) It is assumed that seeds in the separate overlays are transmitting the stream in a synchronous manner. Synchronization is achieved in AnySee by using NTP (Network Time Protocol) and is used primarily in measurement of delays in the network for formation of the tree-based overlays.

\(^5\) This is done in order to gain a “close approximation” of the physical topology underlying the overlay network [3]. By using paths with minimum delays from the end-host to the source, we can filter out shortest paths (in the physical network) and use them in the overlay network – this is especially advantageous since lesser number of internet resources are utilized in shorter paths than longer ones, and there is a better chance of getting higher data rate in shorter paths than in longer paths (i.e., the longer the path, the more the number of links involved in the physical topology, meaning the higher the probability that any one of those links is congested or being heavily utilized). Although using paths with minimum delays does not guarantee sufficient data rates, it is a better method for selecting neighbors (parent(s) in AnySee) than the random selection of neighbors in Coolstreaming.
In this figure, $S_1$ and $S_2$ are seeds transmitting media streams to all other nodes in their respective overlays – this is shown by the directed, solid edges. Each tree is formed on the basis of an upper limit of 10ms delay.

Considering that each node in the above network has only a single path to its seed, a parent node’s failure would drastically affect its child’s playback continuity – it may even require the child to re-buffer the stream from a separate seed from scratch (which, as mentioned above, takes about 40 seconds). In order to cater for this possibility, nodes actively participating in one overlay may have backup paths in other overlays – in the above figure, node D and G’s backup paths from $S_2$’s overlays are shown by directed, dotted edges (only two backup paths are shown for clarity’s sake). If node D (G’s parent) were to fail or abruptly stop/slow streaming to G, G may switch to its backup path hence maintaining playback continuity to whatever extent possible.

As can be expected, AnySee provides higher playback continuity than Coolstreaming. It also tends to accommodate for the large failure-recovery times exhibited by tree-overlays by introducing its concept of active and backup streams. However, it is assumed that all seeds transmitting one media event (such as Larry King Live) do so in synchrony – i.e., no seed lags behind the other in its transmission of the media stream. In reality, this may not always be the case – seeds may be transmitting different parts of the stream at any given time such that one lags behind the other. In such a scenario, switching from one’s active-overlay to its backup-overlay (i.e., switching from the transmission of media from one seed to another seed’s transmission) under dynamic conditions (i.e., events such as node failures) may cause re-buffering of media streams from scratch, lowering playback continuity for the end-user(s)! In the next section, we propose an improvement in AnySee that may provide relief in such a scenario.
Modifying AnySee

The main idea behind our modification to AnySee is the observation that peers, within a tree-overlay, have source-to-end delays that are at most a certain specified upper-limit (for example 10ms). While peers forming an active streaming path in an overlay choose a path with minimum source-to-end delay, other paths that guarantee delay lower than or equal to the upper-limit also exist in the same overlay.

Our observation is that given the aforementioned problem (assumed synchrony of different seeds), it is a better idea to prioritize backup paths within a tree over backup paths in other overlays – i.e., in case of a node failure, we should consider activating a backup path within our active-overlay instead of considering activating a backup path outside our active-overlay. For example, in the following figure, node G has backup paths with E and F in its active-overlay, and a backup path with X in its backup-overlay.

This modification may address the aforementioned problem in case a child node’s parent fails, but only if its parent is not the seeding node itself, in which case the child node will have to activate the backup path with another seed’s overlay. However, seeding nodes are expected to be stable and the probability of intermediate node-failure is higher than the probability of seeding node-failure.
Even though we expect our modification to improve playback continuity in AnySee in dynamic environments, the extent of the improvement is yet to be analyzed.

**Future Work**

In order to perform a comparative analysis of AnySee with Modified AnySee, and to perform a buffer analysis of AnySee, Modified AnySee and Coolstreaming, we shall implement all three approaches using Mace\(^6\). We will test these implementations on PlanetLab to ensure our simulation results match real environments, both stable and unstable. While it shall not be possible to perform tests using actual video streams given the shortage of time, we believe the same effects can be achieved by using randomly generated data on designated seeds.

We expect the results of the simulations will be sufficient for a quantitative analysis of 1) improvements achieved through modification of AnySee, and 2) effect of buffer sizes on playback continuity in stable and dynamic environments.

**References**


---

\(^6\) **Mace** is a language and source-to-source compiler that translates a concise but expressive distributed system specification into a C++ implementation.