Resilient Tree-based Live Streaming in Reality

Benjamin Schiller and Giang Nguyen and Thorsten Strufe
P2P Networks
TU Darmstadt
[schiller, nguyen, strufe][at]cs.tu-darmstadt.de

I. Introduction

Peer-to-Peer (P2P) streaming has been a scalable and cost-efficient solution to deliver live video streams. To reduce the bandwidth demands at video servers, it uses resources (mostly upload bandwidths) of participating peers. However, depending on unreliable end-hosts to build an efficient and robust streaming system is a difficult task. Various approaches have been proposed. They are often classified by the type of overlay topologies and the way video packets are disseminated in the overlay. Single-tree-push systems have low latency, but are less resilient to network dynamics. Mesh-pull systems are more resilient to failure and churn, but suffer larger delays and overhead. Hybrid systems are based on a mesh-pull system. Peers in these systems may switch between pull and push modes to deliver video packets at certain conditions. However, their resistance against Denial-of-Service (DoS) attacks is unknown. Multi-tree-push systems split the video stream into stripes, and deliver them using disjoint trees. The latency is therefore low, which is suitable for a live streaming service. Moreover, it is also proven resilient against DoS attacks via formal analysis and simulations [1]. Nonetheless, it is not known how the system performs in real world conditions.

A deployable P2P live streaming system that is based on the multi-tree-push approach would help shed light to this question. Moreover, this can help us to obtain more realistic estimations on latency, understand how the system behaves in an Internet-wide deployment, and have practical insights on the implementation obstacles which were abstracted in the simulation. The task, however, is not straight-forward. On one hand, the system has to be comprehensive to cope with various sophisticated situation on real-world networks. On the other hand, it should not couple tightly to a specific class of devices.

Our main contribution in this work is a deployable multi-tree-push system for P2P-based live streaming. It runs on both desktop PCs and Android-based mobile devices. Additionally, it provides controlling, monitoring, and measurement functionalities which help with debugging in the development phase, visualize the topology during a demonstration, and support the deployment of test scenarios in a distributed setting. Besides, the generic architecture of the system also allows for the extension to other classes of streaming systems.

II. Streaming System

We developed a multi-tree-push streaming system which is resilient to failures and attacks. In contrast to single-tree systems, failures in one of the trees can be compensated by using forward error correction or special video codecs. Nodes only know their parents and children in the tree topologies. Thereby, an attacker cannot easily collect topological information about the network and hence can not identify the source or other important nodes of the topology.

Based on their local knowledge only, peers optimize the topology whenever their available upload bandwidth differs from their current usage. In case a peer has more children than it can serve, it pushes a child down the tree, i.e., it asks one of its other children to overtake the parent role. Whenever a peer has enough capacities available to upload to another child, it pulls up a peer, i.e., it asks a child to hand-over one of its children.

We implemented three strategies to decide which peer to pull up or push down: random, bandwidth, and location. As a baseline, peers using the *random* strategy select children to push down or pull up randomly. In the *bandwidth* strategy, peers with high bandwidth are placed closer to the source. Using the *location*-based approach, topologically close peers are clustered based on an approximation of their network position.

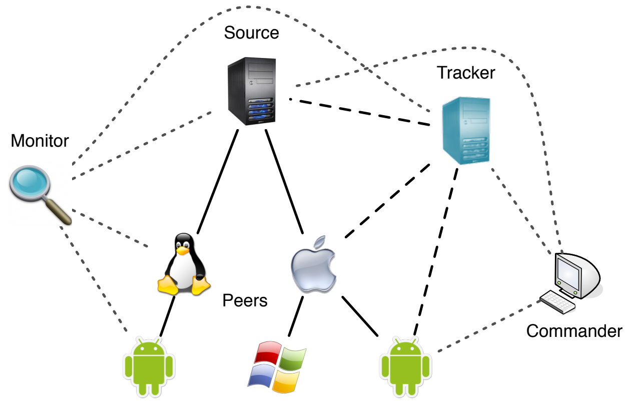

Fig. 1. Components of the streaming system - solid lines indicate topology maintenance and payload traffic, dashed lines represent bootstrapping traffic, and dotted lines show measurement traffic

The streaming system consists of three core components: source, tracker, and peer (cf. Figure 1. A *source* is the initial provider of a live video stream. It is the root of all trees used for the push-based distribution of the data in each stripe. The *tracker* maintains a list of all sources and all peers currently connected to the respective overlay. *Peers* are the clients that want to receive a specific video stream. After obtaining some bootstrap nodes from the tracker, they join the overlay, become part of each stripe's tree topology, and participate in the distribution of the video stream.

For demonstration purposes, the *monitor* component allows for the surveilance of a running system. By collecting topology information and runtime statistics from sources and peers, it facilitates live monitoring of the system's topology and the properties of its components. While it is interesting to observe the properties and topology changes of a live system, it is also crucial for developing new optimization strategies in such a complicated, distributed system.

Furthermore, the streaming system is enriched by the *commander* component. It enables the setup and evaluation of scenarios in a distributed test environment. As input, the commander takes a scenario description that defines the course of events such as peers joining and leaving the system or crashing unintentionally. This allows for the evaluation of various churn scenarios as well as attacks on the system. After executing such a scenario, the commander collects a large set of measurements from all components that can be evaluated afterwards.

III. Implementation

The architecture of the streaming system (cf. Figure 2) has been developed in analogy to our simulation model for p2p-based streaming systems [2]. The peer and source components both provide a streaming layer supplying all parts required to distribute the video stream between the entities. The *packet generator* creates packets from an input video stream. The *buffer* is used on the peer side to buffer incoming data which is then passed down to the *video player*. The task of the *connection maintenance* part is to build the streaming overlay and keep it connected while the *topology optimization* performs optimizations of the overlay. Both directly interact with the *bootstrap* part of the discovery services that allows peers to discovers others by querying the *membership management* of the tracker.

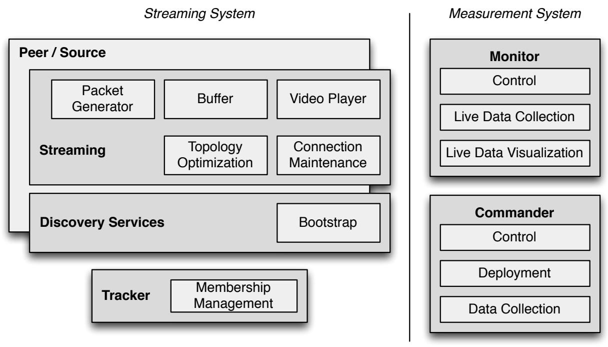

Fig. 2. Architecture of the streaming and measurement system's components

The measurement system (cf. Figure 2), consisting of monitor and commander, provides capabilities to *control* the behavior and configuration of peers and source during a demonstration or a scenario evaluation. The monitor allows to *collect live data* in a running system and also *visualize it*. In contrast, the commander provides means to *deploy* a large-scale test in a distributed system and *collect data* after its execution.

All components of our architecture can be replaced by other implementations. This allows for the evaluation of different optimization techniques, buffer strategies, and even different overlay topologies.

Currently, streaming and measurement system are implemented in Java. All components can be run on desktop machines (Windows, Mac, Linux). Source and peer are also available for Android devices. The packet generator creates an RTP stream from an input video stream which can easily be distributed among any number of stripes. On all platforms, VLC is used to display the received video streams.

The code is open source and licenced under the GPL.

IV. Planned Demonstration

In the demonstration we plan to show the following features:

a) Streaming system: Our implementation of a multi-tree live streaming system which works on desktop machines (Windows, Mac, Linux) and mobile devices (Android).

b) Monitoring system: A monitoring systems to visualize the topology of the running streaming system and modify the behavior of nodes (cf. Figure 3.

c) Optimization strategies: Impact of different optimization strategies on the overlay topologies as well as performance measures.

d) Attack resilience: Showcasing different scenarios, including attacks on the participating nodes, to show that the system is actually resilient to attacks.

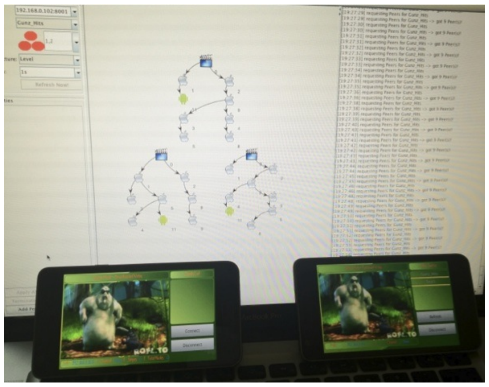

Fig. 3. Two peers on Android devices and a monitor on a desktop machine

The audience will be able to download the peer application to their Android devices, select an available stream, and join the overlay of the desired overlay.

References

[1] M. Brinkmeier, G. Schafer, and T. Strufe. Optimally dos resistant p2p topologies for live multimedia streaming. *IEEE Transactions on Parallel and Distributed Systems*, 20(6):831 –844, june 2009.

[2] G. Nguyen, M. Fischer, and T. Strufe. Ossim: A generic simulation framework for overlay streaming. In *Summer Computer Simulation Conference*, 2013.