

WineStreamer(s): Flexible P2P-TV Streaming Applications

L. Abeni², Á. Bakay⁴, R. Birke¹, G. Ciccarelli², E. Leonardi¹, R. Lo Cigno², C. Kiraly², M. Mellia¹,
S. Niccolini³, J. Seedorf³, T. Szemethy⁴, G. Tropea⁵

¹Politecnico di Torino, Italy; ²University of Trento, Italy; ³NEC Laboratories Europe, Germany; ⁴Netvisor Ltd, Hungary; ⁵LightComm s.r.l, Italy

I. DEMO SCENARIO

The fraction of Internet traffic guzzled by P2P-TV systems is increasing steadily and commercial systems are widespread¹. Research is not lagging behind and scores of interesting contributions appear every year in the literature.

Within this scenario the NAPA-WINE EU project² has contributed novel ideas, models, algorithms and simulations³, but also Open Source, running software. This includes standard C libraries of building blocks that can be used by the community to experiment with novel techniques, as well as to build fully fledged P2P-TV systems, distributing on-the-air TV channels in real time to large swarms. NAPA-WINE is also one of the major promoters of the ALTO⁴ framework for supporting the interaction between network operators and P2P applications.

This demo shows, live over the Internet, the impact of different system choices and algorithms on the streaming performance, and on the network impact of the P2P-TV system. The software used is WineStreamer, the latest version of the distribution platform developed by NAPA-WINE, coupled with a repository and visualizer that enable to monitor the swarm performance in real-time.

II. WINESTREAMER ARCHITECTURE

Fig. 1 presents the functional architecture of WineStreamer, which is based on chunkization of the media and swarming on a generic mesh overlay. WineStreamer is one of the incarnations of PeerStreamer⁵, the more general P2P-TV application we developed with many different versions devoted to both research and ‘production’. WineStreamer is organized in functional modules as described below.

Content Playout manages media formats and rebuilds the stream after de-chunkization. It also includes a QoE estimator based on a neural network (see Sect. III-C). At the source, a similar module performs encoding and chunkization.

Overlay Management includes peer discovery and sampling, as well as a Topology Manager that implements more sophisticated techniques than simply building a randomly sampled

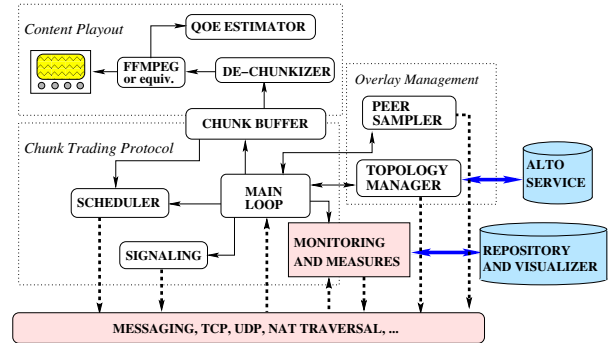


Fig. 1. The functional architecture of a Napa-Wine P2P-TV client.

topology. It can include both network awareness based on local measures and interaction with an ALTO server to obtain information that cannot be accessed locally (e.g., link costs).

Chunk Trading Protocol handles the diffusion of the stream. The main loop of the WineStreamer is within this functional block, dictating timing and coordinating the other modules. Further, specific signaling and scheduling algorithms are present as sub-blocks. In particular, WineStreamer implements an offer/trade protocol in which peers send *offer* signaling messages to some of their neighboring peers, which explicitly accept the chunks before transmission, reducing duplication to zero. Chunks and peers are selected based on scheduling algorithms that can be tuned to the system designer needs (see Sect. III-A) with the aim of meeting performance goals while minimizing the network footprint.

Monitoring & Measures is a specialized module interacting with the Messaging through callbacks in order to enable passive measures (RTT, the number of hops, capacity, available bandwidth, ...) that can be used by the Chunk Trading Protocol as well as the Overlay Management to implement network aware strategies. Some metrics are also periodically published to the external repository and swarm visualizer, which empowers real-time monitoring of the entire distributed application (see Sect. III-B).

Messaging is an abstraction of the networking interface hiding networking details to the other modules. It includes functions like NAT traversal, shaping (if required), peer naming, chunk and message segmentation and reassembly, etc. as well as a callback structure to enable passive monitoring functions.

The streaming architecture described above has been implemented based on a set of generic libraries (GRAPES - Generic Resource Aware P2P Environment for Streaming) available to the community for development of P2P streaming applications.

This work is supported by the European Commission through the NAPA-WINE Project (Network-Aware P2P-TV Application over Wise Network – www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412

¹See for instance <http://pplive.com>, <http://www.soapcast.com>, <http://www.tvants.com>, and many others.

²See <http://www.napa-wine.eu> for details

³<http://www.napa-wine.eu/cgi-bin/twiki/view/Public/DocumentsAndPapers> collects the contributions

⁴“Application-Layer Traffic Optimization (ALTO) Problem Statement,” IETF, RFC 5693, Oct. 2009.

⁵<http://peerstreamer.org>

III. DEMO STRUCTURE

This demo shows the performance and characteristics of WineStreamer when different choices and decisions are taken. The demo is based on the existence in the Internet of several different distribution swarms with different characteristics, to which we can locally connect and show the different behavior. Some parameters can also be changed in run-time in some swarms showing the change in behavior and performance. In the following we detail some of the features that are shown in the demo.

A. Network Aware Topology and Scheduling

One of the key features of NAPA-WINE streamers is being *network aware*, which means having the capability of adapting to network scenarios and conditions, maintaining the video quality while posing a gentle footprint on the network. Network awareness comes in many different flavors, from ISP-based cooperation (see Sect. III-B) to dynamically modify topology and scheduling behavior based on network-level measurements. As an example, Fig. 2, reports the percentage of the average amount of bytes downloaded by each peer from neighbors at a given RTT with WineStreamer when the application is endowed with delay awareness or not, it is clear that the RTT awareness reduces significantly the amount of bytes transferred from far away locations. During the demo the effects of other network parameters like uplink bandwidth will also be shown.

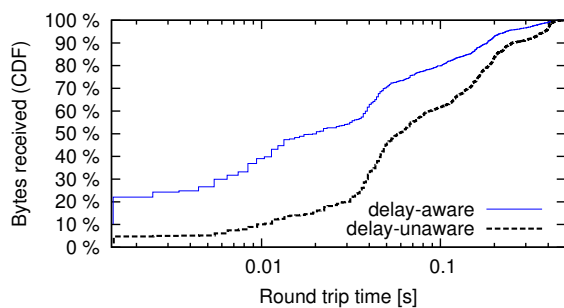


Fig. 2. Average bytes downloaded by each peer from neighbors at a given RTT with and without delay awareness (PlanetLab experiments)

B. ALTO Support

The effects of different ALTO peer selection strategies are shown *live* with the NAPA-WINE *SwarmVisualizer* tool. Examining the traffic flows among peers with the *SwarmVisualizer* allows the visualization of the effect of ALTO-based peer selection on the overlay topology. As an example, the upper plot in Fig. 3 shows traffic flows exceeding a specific threshold (e.g., an average download rate within a certain period) for a P2P live streaming swarm using ALTO, clustering peers from the same geographic location/area. The lower plot in Fig. 3 displays a swarm not using ALTO for the same settings. The demo shows to what extent traffic localization is possible with ALTO while still achieving low chunk loss and delay (as demanded by live streaming applications).

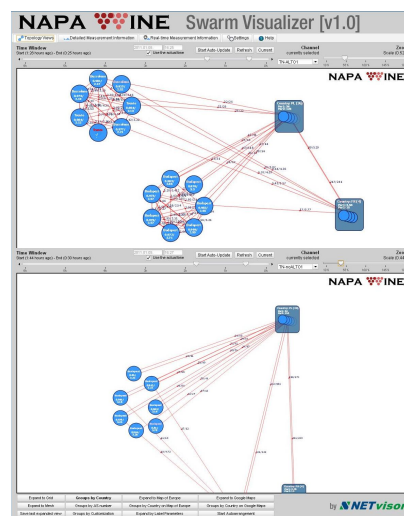


Fig. 3. P2P Swarm with ALTO Support (upper) and without it (lower)

C. QoE monitoring

The QoE estimator is based on a neural network, trained with synthetic loss and delay traces, which estimates audio and video quality through the estimation of the time average of the PSNR (Peak Signal to Noise Ratio). Details on both the neural network and PSNR estimation are beyond the scope of this abstract. During the demo, the actual matching of the estimation with the users' perceived QoE will be shown. The presence of the estimator enables feedback to the streaming system in the node and in the entire swarm in order to take countermeasures. Fig. 4 reports as an example of the capabilities shown in the demo, the table at the left shows estimated PSNR as a function of the parameters used for the estimation, which are: *i*) the burstiness of frame losses (number of consecutively lost frames), derived from the playout queue density Q_d defined as the number of frames in the playout buffer divided by the total number of expected frames (i.e. index of oldest frame available - index of most recent frame available), and *ii*) the number of frames per second missing at playout M_f . The screenshot on the right hand side of the figure, of awful quality, illustrates that the estimated PSNR (28.9, last row of the table in bold) is a good estimation of the quality degradation.

Q_d	M_f	PSNR
100%	0	38.6
96%	1	36.3
82%	4	32.4
63%	9	30.1
20%	14	28.9

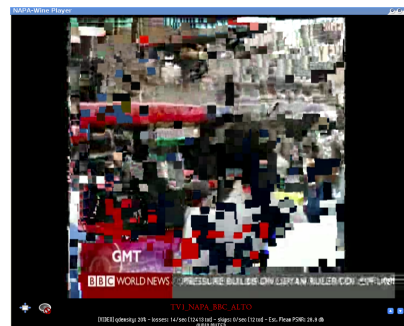


Fig. 4. Estimated PSNR values (table) in different playout buffer conditions; the last (bold) line correspond to the picture quality on the right.