Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 5 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

PARTYHUB - A PEERSTREAMER CONFERENCE APPLICATION

Luca Baldesi, Leonardo Maccari, Renato Lo Cigno

November 2013

Technical Report # DISI-13-037

PartyHub

A PeerStreamer Conference Application

Technical Report DISI-13-037

Luca Baldesi*, Leonardo Maccari*, Renato Lo Cigno*
*Dept. of Information Engineering and Computer Science, University of Trento, Italy
luca.baldesi@unitn.it, locigno@disi.unitn.it, maccari@disi.unitn.it

***Abstract*—Video conferencing is still a lively research field, and for both personal and business applications it is reasonable to expect that the interest in conferencing services will grow in the future. However, available tools lack of scalability and flexibility features and the more widespread ones have proven to be a possible threat for our privacy. This technical report focuses on the design and development of PartyHub, an open source peer-to-peer conferencing platform, entirely customizable, distributed and privacy aware. This goal can be achieved porting a peer-to-peer live streaming platform, PeerStreamer [1] to the video conferencing context. In this report we document the test performed in order to verify the suitability of this idea and the future steps and challenges needed to develop PartyHub.**

I. Introduction

Video conferencing is getting more and more popular as tool lying between the telephone call and the face-to-face meeting. Because of its potential it can be used both in the work and in the home environments. Since it mainly represents "*the closest thing to being here*", it can be used for almost every aspect of our daily communication.

Since a few years, some platforms are capable to perform video conferencing. The most known, among the others are *Google Hangout* and *Skype*. Those platforms offer a conferencing service with some restrictions, in fact both services allow up to ten contemporary users in a session. Furthermore Skype application requires a premium account (about 3.50€ per month) to enable video conferencing among more than two users.

Furthermore the way these services operate is not clear since there is no access to source code and the centralization of the users' data could bring to unwanted operator behaviors, like users activity tracing.

In contrast with these centralized applications other technologies can be exploited to offer the same service. In particular the peer-to-peer technology can be used to obtain a more scalable and decentralized system that would overcome the limits imposed by proprietary platforms and the presence of a *big-brother*.

Our proposal is focused on developing and deploying a complete peer-to-peer video conferencing application based on the PeerStreamer platform[1] called *PartyHub*. This technical report presents the results obtained testing the PeerStreamer libraries in a video conferencing context.

[1]http://peerstreamer.org

The rest of the paper is organized as follows. An overview on the main drawbacks of traditional video conference tools is given in Section II, while Section III presents the target service architecture and the platform used for the test.

Results are reported in Section IV. Finally, Section VI concludes the document.

II. Conference tools

The most popular video conferencing tools like Google Hangout and Skype are closed source, which means that is very difficult to study their behaviour. Only few scientific articles [2] (most of them outdated) try to analyse them using indirect measurements.

The closed source nature combined with the related patents totally preclude any possible improvement by the users. Furthermore that leads to an unpredictable manipulation of users' data which could, in general, be traced and stored for commercial (and profiling) purposes, as it has been shown in the recent PRISM scandal [3].

Other tools such as [4] are based on the *Multicast Bone* [5] and require particular hardware and software in order to operate across the Internet.

Tools like *BigBlueButton*[2] and *OpenMeetings*[3] are open source web based conferencing tools but they are strongly centralized. Consequently they are resource consuming in terms of bandwidth and storage space and they cannot scale to a large number of users without costly resources.

III. A new paradigm

The use of peer-to-peer technologies for video conferencing application has not been well addressed so far in the literature. The scalability of this approach can overcome the problems related to the lack of resources generally limiting the number of parallel users. Given that the video data delivery respects some real-time constraints (discussed in the Section III-A) the overlay composed by the peers permits to share their contents following the cheaper paths among them. Furthermore, it does not require a centralized server which means that anybody could set up his own peer overlay without the need of particular hardware or bandwidth resources. As a consequence there is also an intrinsic privacy advancement compared to centralized approaches.

[2]http://bigbluebutton.org/
[3]https://code.google.com/p/openmeetings/

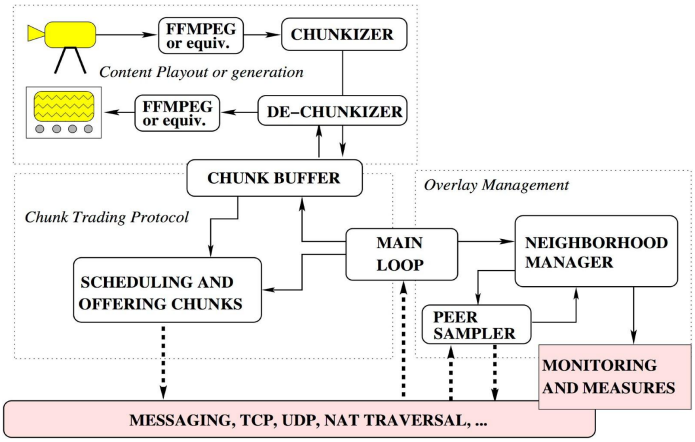

Figure 1. PeerStreamer architecture.

Obviously, the choice of open source implies that the platform will be easily extensible and that the community of users can customize it to fit its needs.

A. PeerStreamer

PeerStreamer is an open source platform for live peer-to-peer video streaming. It has been developed as part of the European Napa-Wine project and it is capable of exploiting the peer-to-peer mesh overlay in order to address the delivery of media contents optimizing the receiving delay. PeerStreamer is composed of different customizable parts. It takes great advantage from the GRAPES [6] toolkit library, which has also been developed as part of Napa-Wine, and implements the required modules that a live peer-to-peer application needs.

Each of the GRAPES component is greatly customizable, even with respect to the algorithms involved in the peer-to-peer overlay management.

Much of the interest around PeerStreamer depends on its capacity of scaling over thousands of peers keeping a very low latency in the stream packets delivery. Using the simulation platform PlanetLab [7], some experiments have been performed and the results shows that the delivery delay of each packets grows only logarithmically with the number of peers. In practice even with more than two thousands peers the transmission only requires a few seconds to reach the farthest peer. In the television broadcasting context, this can be easily defined as *live*.

This important feature is achieved through the epidemic data diffusion scheme implemented in PeerStreamer. Following this scheme, the data source will inject in the overlay only few copies of each packet. The receiving peers will take care to redistribute those packets to other peers until every peer have received its own copy of the packets [8].

The desirable features of a live streaming application are:

- spread the contents from one peer (source) to the others;
- transmit the packets at a (almost) stable rate;
- receive as much packets as possible in order.

Other peer-to-peer applications are designed to maximize the expected throughput and are not suitable for live streaming due to the time constraints of the overall transmission on the overlay. PeerStreamer instead is designed to minimize the receiving delay.

PeerStreamer and GRAPES include algorithms capable of addressing the time constraints and the management problems related to a stable, best-effort, peer-to-peer live application. The HRC algorithm [8] has been specifically designed to obtain the maximum aggregated throughput for a node belonging to a peer-to-peer overlay without increasing the delivery delay. In practice it realizes a congestion control driven by calculating the delay the packets spend in the transmission queue. In this way throughput is maximized while packets collisions are minimized. Even the topology managing has been optimized to reduce the delivery time of the video packets. In [9] it is shown that the algorithm is capable of exploiting the peers with the highest available bandwidth, keeping them as close as possible (in the overlay hops meaning) to the source. This makes it possible to reach a broad content diffusion with stringent time constraints.

Other PeerStreamer features are also available:

- the ALTO support;
- fast and efficient code;
- media awareness;
- active and passive monitoring;
- smart scheduling of peers and video chunks.

Media awareness can be exploited in order to deliver meaningful video packets. In fact, not all the frames of a video stream carry enough information needed to display a complete video image. Keeping that in mind PeerStreamer forges packets with at least one *anchor frame*, granting that the loss of a packet will not affect the correct interpretation of the subsequent ones.

The process of forging these *smart* packets, called *chunks* is performed by the *chunkizer*. This and the other components of PeerStreamer are depicted in Figure 1.

PeerStreamer can be logically split in three main components; the *streamer* which is the responsible for setting up the peers overlay and spread the contents across the network, the *source tools* which injects the media contents in the overlay, and the *player application* which simply takes care of taking the video parts from a streamer and display them to the user. The resulting PeerStreamer logical overlay is depicted in Figure 2.

IV. Proof of concept

In order to verify the suitability of the PeerStreamer platform for video conferencing a simple test has been set up. In the tested scenario, three PeerStreamer overlays have been overlapped, so that three peers were acting both as source and as content consumer at the same time (Figure 3).

The scenario addressed in this test relates to three users willing to communicate through video conferencing. The three users

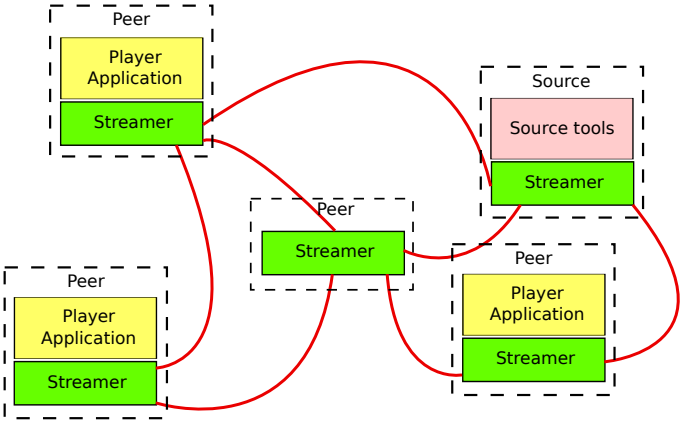

Figure 2. PeerStreamer network architecture. The peer-to-peer overlay is composed of the streamers; the source tools inject the video packets in the overlay and the player applications perform the playout. Not all the peers need to implement the player or the source.

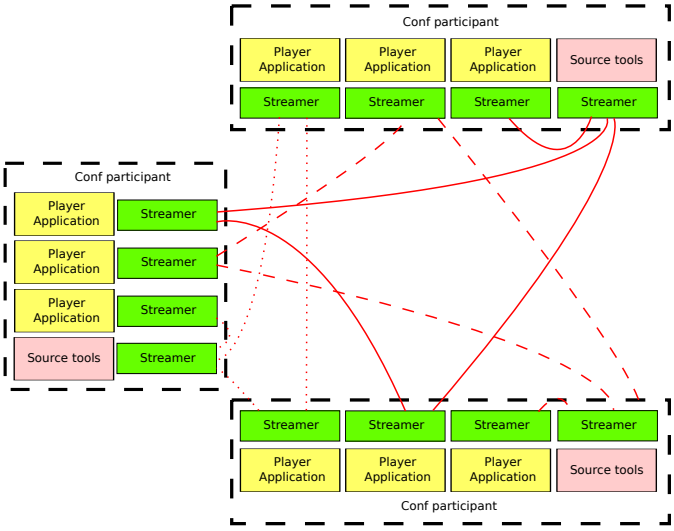

Figure 3. PeerStreamer based three-way conference. Each of the participants offers and consumes the video contents.

set up their own PeerStreamer source and join the overlay of the others.

In order to realize this test a simple architecture has been set up. All the users information was hard coded as well as the required commands to launch the PeerStreamer instances.

The steps performed by each single conference participants are the following:

1) acquire the video content to inject in the user overlay;
2) launch the PeerStreamer source, combining the source tools and a streamer instance;
3) launch a player application and a streamer instance in order to display the content injected by its own source;
4) launch two other istances of the streamer and the player application in order to join the other users overlays and display their video contents.

The result of the execution of this test scenario is that the three users can simultaneously access all video contents injected

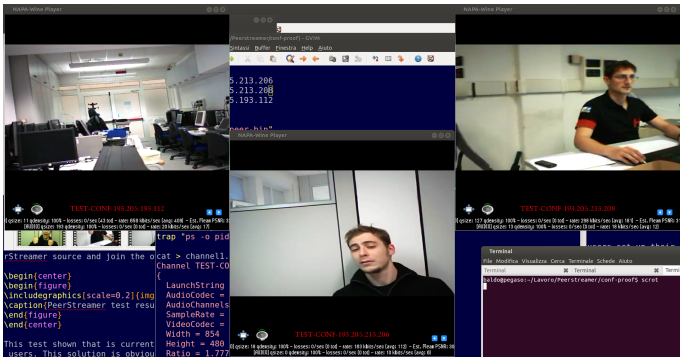

Figure 4. PeerStreamer testing. On each of the users screens there is the simultaneous playout of the videos.

in the peer overlays: the one they are actually injecting and the ones obtained from the other peers.

A screenshot of the resulting interface is reported in Figure 4.

This test shows that setting up overlapped PeerStreamer overlays in order to realize a three-way communication among different users is already feasible. This solution is obviously non optimal due to the additional redundant managing and signalling overhead generated by different overlays among the same peers. Nevertheless PeerStreamer has shown to easily support the increased network activity and keeping the overall cpu load at low levels on standard desktop machines.

V. Open Issues

Although PeerStreamer grants high flexibility, low-delay content delivery, and it has proven to be suitable for basic video conferencing, some work has to be done in order to port its code to the fully-featured video conferencing tool PartyHub.

In fact, even if PeerStreamer grants low-delay content delivery with overlay of thousands of peers, a video conferencing application requires even lower delivery delays, in the order of hundreds of milliseconds.

A. Conference oriented overlay

Currently, a multi-source PeerStreamer communication involves the use of overlapped peers overlays. As already reported in Section IV this behaviour is neither efficient nor easily manageable. In fact, part of the signalling communication is replicated for each overlay and could be aggregated instead. Moreover, specific data belonging to a source can currently be exchanged only in its own overlay even if it could be more convenient to enable cross overlay data exchange. One possible strategy to address this specific problem may be to have one signalling overlay with multiple data sub-overlays.

B. Specific Topology Management

In order to further reduce the PeerStreamer delivery delay, topology algorithms and packets delivery policies need to be further studied. For instance it is common to have peers with different uplink bandwidth capacity. For this reason the

topology algorithm could be designed to fairly redistribute the packet transmissions among the users; if a peer has poor uplink capacity it should send only few packet copies directly to the more bandwidth capable peers which could then redistribute them exploiting their uplink capability. That should lead to a fair redistribution of the overall uplink bandwidth resulting in the same expected transmission delay per peer.

C. Quality Management

Another key problem related to the specific application we are addressing regards the inactive users. Typically, in a conferencing scenario, there is one active user, i.e. talking to the others, while the other users are passive, i.e. hearing the speech. In this context all the video contents transmitted by the passive users is much less valuable compared to the video contents of the active user. So it is worth to study and implement a method to decrease the quality of video contents belonging to the passive users injected in the overlay in order to save bandwidth and computational load with the overall result of a better quality of experience.

D. Web Interface

The currently available PeerStreamer interface was not designed as a conferencing tool. A possible solution could be the designing of a web-based interface which has been proved to be generally more appealing for the users. Moreover it could grant a higher degree of flexibility, both for the choice of the video player plug-in used to play out a video stream and for the possibility of reshaping the interface to embed as many video plug-in as needed by the number of users participating in the conference.

The choice of a web interface could be the first step in designing the PartyHub architecture. A solution currently under evaluation could be orienting the control communication between the conferencing platform components over the HTTP protocol. Representational State Transfer (ReST) is a lightweight and efficient architectural style which could well suite PartyHub needs. Furthermore, ReST paradigm grants an high degree of interoperability with other web applications.

Following the test scenario it is possible to identify three main components for the conferencing tool: the GUI, a controller responsible of driving the streamer instances of a conference participant and a third-party service capable of providing the synchronization info among the users.

A ReST oriented architecture is then sketched in Figure 5. The users participating to the same video conference are identified by the concept of *group*. Each user keeps a daemon running on his host (the streamers controller), responsible of launching and managing the streamer instances, and uses the web GUI to drive, through HTTP transactions, the streamers controller, getting the access to a specific group overlay video contents. The contents are then displayed through as many playback objects as needed (by the number of the group users) inside the web GUI.

A separate centralized service, the *group manager* will store all the information related to a specific group of users. In particular it is responsible to provide the required information to access the users overlays. When needed, the streamers controller can contact the group manager through HTTP transactions to obtain these information and then launch and attach a streamer to a specific user overlay.

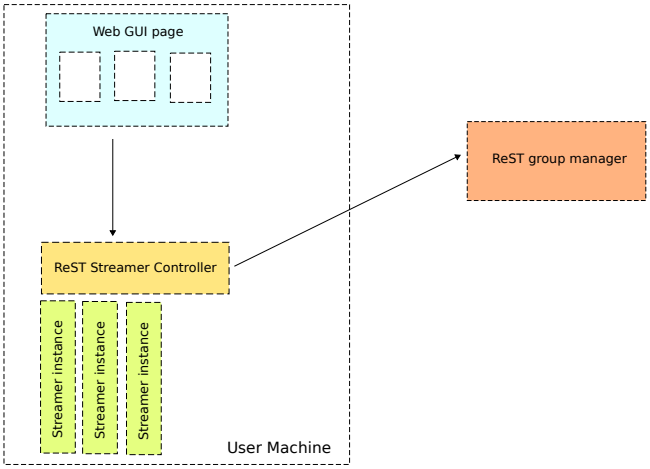

Figure 5. Possible PartyHub architecture.

Using a web interface would introduce also the possibility to easily mesh-up and integrate other web-based services with PartyHub. For instance it could be possible to add *social* features like a messaging chat shared among the users or introduce the possibility of showing commercial banners.

E. Security Aspects

All the data exchanged between PeerStreamer users is currently not cyphered. For a public content broadcaster it is not a problem but this feature is needed for personal intercommunications. Different techniques can be applied in order to provide security and privacy features. As a basic measure, a centralized registration on the group manager could be used to generate a shared key to encrypt the video conference traffic. Alternatively, a system based on public/private keys and a web of trust could be exploited, with an initial distributed negotiation among the peers.

F. NAT and Firewall Traversal

Currently PeerStreamer requires that every source could be reachable through a public IP address. That is reasonable in the peer-to-peer TV broadcasting context where there are a few major content providers with server infrastructure. Instead, in the video conferencing context, all the users have to be reachable by the others even with the presence of NAT or other middle-boxes. PartyHub should include some components responsible of passing through those network hosts with a set of techniques that perform session traversal and hole-punching, some of which are already present in PeerStreamer.

VI. Conclusion

Video conferencing has been a lively research field in the past and today there is still room for innovation and for suitable solutions to the real-time and scalability issues. Besides current popular available solutions, our proposal focuses on exploiting the peer-to-peer technology and the live streaming tools developed during Napa-Wine project in order to provide a novel video conferencing platform addressing the live streaming problems. PartyHub, should be able to instantiate a multi channel overlay in which all the conference participants can inject their video contents and from which they can get the video streams of the others. We performed tests with PeerStreamer in order to verify the feasibility of this idea and realized it is already possible, even if not optimized. PartyHub should thus provide a service with a quality of experience comparable to, or even better than the currently available conferencing software. In addition PartyHub will offer the scalability, load distribution and customizability given by the open source approach.

References

[1] R. Birke, E. Leonardi, M. Mellia, A. Bakay, T. Szemethy, C. Kiraly, R. Lo Cigno, F. Mathieu, L. Muscariello, S. Niccolini, J. Seedorf, and G. Tropea, "Architecture of a network-aware p2p-tv application: The napa-wine approach," *IEEE Communications Magazine*, vol. 49, pp. 154–163, 06/2011 2011.

[2] S. Baset and H. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," *CoRR*, vol. abs/cs/0412017, 2004.

[3] "Washington Post here's everything we know about prism to date," http://www.washingtonpost.com/blogs/wonkblog/wp/2013/06/12/heres-everything-we-know-about-prism-to-date/, date: 2013-06-13.

[4] S. McCanne and V. Jacobson, "vic: a flexible framework for packet video," in *Proceedings of the third ACM international conference on Multimedia*, ser. MULTIMEDIA '95. New York, NY, USA: ACM, 1995, pp. 511–522. [Online]. Available: http://doi.acm.org/10.1145/217279.215315

[5] S. E. Deering, "Multicast routing in a datagram internetwork," Ph.D. dissertation, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1991.

[6] L. Abeni, C. Kiraly, A. Russo, M. Biazzini, and R. Lo Cigno, "Design and implementation of a generic library for p2p streaming," in *Proceedings of the 2010 ACM workshop on Advanced video streaming techniques for peer-to-peer networks and social networking*, ser. AVSTP2P '10, ACM. New York, NY, USA: ACM, 2010, p. 43–48. [Online]. Available: http://mycite.omikk.bme.hu/doc/90768.pdf

[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003. [Online]. Available: http://doi.acm.org/10.1145/956993.956995

[8] R. Birke, C. Kiraly, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "A delay-based aggregate rate control for p2p streaming systems," *Computer Communications*, vol. 35, pp. 2237 – 2244, 11/2012 2012. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2012.07.005

[9] R. Lobb, A. P. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Adaptive overlay topology for mesh-based p2p-tv systems," in *ACM NOSSDAV 2009*, Virginia, June 2009, conference. [Online]. Available: http://www.telematica.polito.it/mellia/papers/fp21-lobbPS1.pdf