Live P2P streaming in CommunityLab: Experience and Insights

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

***Abstract*—Wireless Community Networks (WCNs) are flourishing as a means of providing Internet access, but most of all as an alternative, bottom-up approach to reduce the digital divide and empower the users with control on their network. Video streaming and conferencing are among the most resource hungry and critical networked applications, and their support on WCNs is fundamental, but difficult as Wireless Community Network (WCN) environments are normally less resource-rich than the traditional Internet. This work presents an initial analysis of an experimental activity with PeerStreamer, a P2P video streaming platform, on the Community-Lab, the WCN testbed of the EU FIRE project CONFINE. The results we present shed light on several different aspects, some good, some other less, of video streaming on WCNs. The experiments highlight the feasibility of P2P video streaming, but they also show that the streaming platform must be tailored ad-hoc for the WCN itself to be able to fully adapt and exploit its features and overcome its limitations. On the other hand, the experiments also show that Community-Lab is not yet fully representative of a WCN, and specifically of those that participate in CONFINE.**

I. Introduction

Global and local communications, the Internet for short, are one of the key components of a modern, developed, thriving society. Wireless Community Networks (WCNs), after more than ten years from their first appearance [1] are experiencing a revival and are blooming in Europe and all over the world. The reasons for this interest and success are multiple and span different disciplines: technical, legal, and socio-economic. The economic and social value of the realization of open access local networks was recognized in a more general context that WCNs [2], [3], but it finds its perfect context in WCNs because of their cheap and incremental deployment and their grass-root, bottom-up design, development and management [4]–[6].

In spite of, or maybe because of, their novelty and social appeal, WCNs often raises skepticism on their technical feasibility, if not outright suspicion on their value. Indeed, their distributed nature, and the complexity of a switched wireless network with all the associated technical challenges, make service provisioning on a WCN non trivial. Moreover, the lack of experience on such networks and the fact that they very often "rise and grow" instead of being planned and designed amplify the challenges.

This work was partially funded by the European Unions 7-th Programme for research, technological development and demonstration under grant agreement No. 288535 "CONFINE" – Open Call 1 project "OSPS"

Among the services that are more appealing, but also more challenging to provide, live video streaming both for events distribution and for conferencing cover a very special role. They are bandwidth hungry, they require low and stable latency, they hardly stand packet losses, and they do not tolerate TCP-like Automatic Retransmission reQuest (ARQ) mechanism, because the delay of a single time-out can be more than the stream can admit, not to mention the fact that ARQ mechanisms imply unicast transmission, which in turn means that the source of the stream must have enough resources for all users: The source should relay on resources typical of a Content Delivery Network (CDN) rather than those of a cheap self-installed Wireless Community Network (WCN). Supporting the streaming with an external, cloud-based system (e.g., Google hangout for conferencing) is also ill suited for a WCN. In fact, the external connectivity in WCNs is very often scarce or expensive, either for strictly commercial reasons or for an explicit choice to favour the development and provisioning of in-WCN services rather than making the WCN a mere access network.

Multiple-unicast streaming is out of question on a WCN, since community nodes are not expected to support such a resource hungry scenario. As IP-multicast is typically not supported by nodes and routers, a P2P approach for live streaming is an appealing alternative, which also maps very well with the social and technical approach of a WCN.

The work presented in this paper is strictly experimental and it concerns the early insights and results we obtained investigating the behavior of PeerStreamer[1] running on top of Community-Lab[2]. PeerStreamer, which is the evolution of the streaming solution developed within the NAPA-WINE EU Project[3], is an Open Source fully distributed platform for P2P live video streaming, and it is shortly described in Sect. II-A. Sect. II-B describes the CONFINE[4] EU Project and, with some more detail, the Community-Lab design and environment. The Community-Lab development is still ongoing, and these experiments are the first to have been attempted and run on it with a real and working application. Albeit it is not yet fully representative of the general environment of a WCN, Community-Lab gives in any case a very good flavour of the challenges found to provision a complex service on a WCN.

The contribution of this work lies in the joint analysis of

[1] http://peerstreamer.org
[2] http://community-lab.net/
[3] http://napa-wine.eu
[4] http://confine-project.eu/

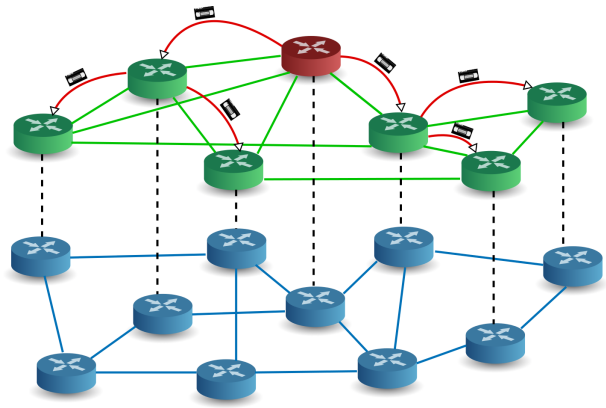

Fig. 1. High level representation of PeerStreamer video distribution process; the source (represented in red) injects a configurable number of copies of each chunk in the overlay (represented in green) and peers complete the distribution adapting to the network resources; in our experiment the source injects a single copy of the chunks.

PeerStreamer and Community-Lab. The results we present give insight in the problems and issues underlying the development of a complex experimental platform, and in the subtleties needed to make it really representative of the real environment. Moreover, the performance of PeerStreamer, the overlay topology it builds, the required adaptations we discuss in the final part of the paper are all novel contributions toward building live video streaming services based on a P2P approach that match the WCN networks spirit and technical development.

Works related to P2P streaming in the Internet abound in the literature, but specific works dedicated to P2P live streaming over WCN are instead, to the best of our knowledge, entirely missing, so that there is actually no "related work" to discuss. A recent PhD dissertation [7] includes some discussion also on P2P streaming on WCNs, but does not touch on live streaming (streaming is intended as Video on Demand (VoD) retrieval), neither presents experiments on real networks.

II. The Environment

The setting of our experiments is the research infrastructure named Community-Lab [8] developed and maintained by CONFINE. On top of this infrastructure we deployed an overlay of PeerStreamer nodes and measured the performance achievable to check if medium/large scale streaming is feasible on such an infrastructure.

A. PeerStreamer background

PeerStreamer is an open source P2P video streaming platform currently supported and maintained by the University of Trento. It is the outcome and follow-up of the NAPA-WINE project and can be considered today the most advanced open and documented platform to build P2P video streaming services.

PeerStreamer, as depicted in Fig. 1, is based on a non-structured, highly dynamic, mesh overlay topology that supports the video distribution process through swarming of elementary information units called *chunks*. A chunk is an arbitrary information unit, but in our experiments we constantly used the mapping of a single video frame per chunk, so as to respect the structure of the video and minimize the encoding delay at the source. The source of the video injects a single copy of each chunk in the overlay transmitting it to one of the peers chosen uniformly at random. The peers complete the distribution selecting chunks and peers for the information exchange following one of many strategies selectable in PeerStreamer when building the actual *incarnation* of the streamer that one want to experiment with.

PeerStreamer is composed of several logical blocks: an I/O module to capture video and reproduce it; a topology management subsystem to build and manage the overlay; a chunk and peer selection (within the overlay) to choose what chunk is to be exchanged with what peer in the overlay. Monitoring tools and a networking abstraction level that helps with NAT traversal and other similar issues (including segmentation and reassembly of chunks to avoid IP fragmentation and selective packet fast retransmission to avoid losing an entire chunk because one packet was lost) complete the logical architecture of the system. Besides the information on PeerStreamer home page, a high level description of the logical architecture of the system can be found in [9], while [10] reports the software architecture of the core libraries, which guarantee high portability and efficiency of the system.

The overlay topology is build with a network aware approach as explained in [11]. In our experiments, however, the topology management has a marginal role, as the very limited number of nodes available collapses the topology basically to a full mesh.

The order in which these chunks are selected is instead generally important as it allows peers endowed with a lot of resources to contribute more, and it also allows a peer to isolate other peers that are not able to contribute useful chunks to the peer itself. PeerStreamer includes different selection strategies including optimal ones derived from [12]; however, during our experiments on the Community-Lab we use a random chunk selection for the sake of results interpretation, as our main goal is understanding the behavior of the WCN level distribution process. The chunk exchange protocol is based on a Push/Pull approach with Offer/Select and Confirmation [13], [14], which ensure that no duplicated chunks are received by any peer and that network resources are well exploited without building long transmission queues that would increase the chunk delivery and consequently also the playout delay.

B. CONFINE and Community-Lab

One of the objectives of the CONFINE IP EU Project is to support the research on WCNs and foster their growth. Community-Lab is the testbed being deployed by CONFINE to this purpose. It is an overlay network ideally built on top of the WCNs that participate to CONFINE, namely Guifi.net[5], FunkFeuer[6] and AWMN[7], respectively in Spain, Austria and

[5]https://www.guifi.net/
[6]http://www.funkfeuer.at/
[7]http://www.awmn.net

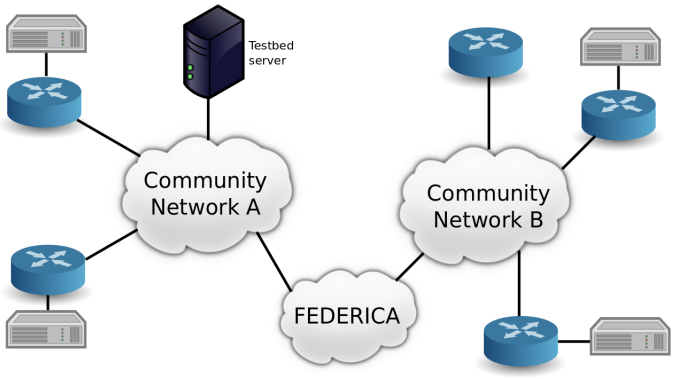

Fig. 2. Community-Lab architecture; blue nodes represent the community nodes while the gray boxes represent the testbed nodes.

Greece. The three networks are federated using the network infrastructure provided by FEDERICA[8].

The Community-Lab is a network of *"research devices"* co-located with a subset of the nodes in the three WCNs. Research devices are not directly part of the WCN, but are devised to provide an isolated environment for experiment, ensuring that the normal WCN operation is not jeopardized. Each device is a standard PC that researchers can access via different interfaces; it generally doesn't have a dedicated wireless interface, but it's interconnected through a local interface with a real node of a WCN. In each research device the researchers can run their own applications, which exploit the overlay defined on the WCNs to communicate. The final goal of Community-Lab is to provide a realistic scenario for researchers to test applications as if they were running on a real WCN.

To help the automation of the experiments, Community-Lab provides a testbed server that orchestrates the research devices and is accessible by researchers. The server can be used to pilot the research devices using a REpresentational State Transfer (REST) web Application Programming Interface (API); Fig. 2 depicts the logical architecture of Community-Lab.

Besides being connected using the WCNs, the research devices are also reachable using a dedicated Virtual Private Network (VPN) that is used for management. The researcher can use the server to control the nodes via the VPN in order to avoid problems due to addressing, Network Address Translation (NAT), etc. The researcher's computer, can be added to the VPN too in order to have direct access to the research devices.

III. Experiments Setup

Community-Lab supports the experiments of multiple researchers using the Slice-based Federation Architecture [15]. The testbed nodes (represented in gray in Fig. 2) are called *research devices* and are an actual part of the WCNs. Research devices are host machines with virtualization capabilities, wired to a community node. Each researcher has access to a certain number of research devices, forming a *slice* available for his experiments. The researcher can choose to have a slice contained in a single WCN or spanning across multiple ones. Each research device can allocate a number of virtual machines. In the Community-Lab terminology, each virtual machine is called a *sliver*. A sliver is the host in which the researcher can run his own software. The same research device can belong to two slices referring to two different experiments, in that research device each experiment will run on a different sliver. Via the REST web API it is possible to query the status of research devices, slices and slivers. The typical workflow to perform experiments with the Community-Lab testbed comprehends the following steps:

1) Create an account on the web controller[9];
2) Log in the web controller and create a new slice;
3) Select the research devices that belong to the slice and create a sliver for each of them;
4) Customize the slivers as needed;
5) Start the slice.

Each sliver is accessible using two addresses, the first is an IPv4 address in a subnet shared by all the slivers and routed via the WCNs and FEDERICA. The second is an IPv6 address reachable via the management network and the VPN. While each couple of slivers can communicate using the IPv6 network, IPv4 connectivity is not guaranteed; it depends on the condition of the underlying WCNs. After the slice creation, researchers should add their research computer to the CONFINE management network and start to interact with their slivers, for instance running programs and collecting logs. The web controller offers an easy guided way to perform this operation.

A. The way experiments are managed

At the time of writing, the Community-Lab is still a work in full progress so that a special effort is needed to avoid some pitfalls. This paper wants to describe and share our experience and also some of the tools we used to perform the experiments. First of all, we developed a simple Bash framework and made it freely available on the web[10] to perform experiments within the Community-Lab. The goal of the framework is to automate actions needed to perform experiments, so it interacts with the web controller and directly with the appropriate sliver. The framework is based on atomic experiments, each experiment is defined through a set of variables such as the slices to be used and the executables to be run on each sliver. Provided an initial configuration file, our system exposes these command interface to the researcher:

- `start`: it collects the slivers addresses of the indicated slice, filter them accordingly to the configuration, upload in each sliver the chosen executable and the configuration files, and start their execution.
- `stop`: to stop the executable execution on the slivers.
- `status`: to check the slivers status, i.e., if they are reachable, responsive and are running the executable.
- `command <cmd>`: to run a remote command on all the slivers.

[8]http://www.fp7-federica.eu/

[9]https://controller.confine-project.eu/

[10]http://halo.disi.unitn.it/baldesi/PublicGits/confine_test_scripts.git

- `retrieve <file>`: to retrieve files from all the slivers;
- `clean`: to stop the slivers and delete all the experiment files (on the slivers and locally).

This framework also enables the preparation of the slivers for the experiments and troubleshooting some problems we met during the tests. For instance, we realized that not all the slivers can communicate using the IPv4 network, this might be due to the WCNs, to the poor connection among the networks, or to any factor out of the researcher control, so the Bash framework allows restricting the experiment to the devices suitable for the experiment. Another problem we encountered is that some of the slivers are not time-synchronized and Network Time Protocol (NTP) cannot run on them. Since the time in each sliver depends on the system time in the research device we could not synchronize the slivers. When running and evaluating real-time system like live streaming this is a non-marginal impairment. For the time being we got around the problem getting the difference from a common time source and rescale our logs using the correct time difference.

B. Experiments performed

Our experiments are composed of ten iterations of identical runs. Each run lasts ten minutes and between each run, slivers are stopped for five minutes. We have to use only slivers belonging to the Guifi.net WCN due to the aforementioned issues, for a total of 16 slivers, far less than what we would like to test, but still meaningful for the experiments and also for small-scale service provisioning.

During our experiments, PeerStreamer uses a low bitrate: 300 kbit/s not to stress the WCN resources. We run experiments in which the PeerStreamer source injects only one copy of each video chunk in the overlay. This is a very challenging scenario, since with the resources of one single video stream we create a whole distribution system. During each run we collect information both about the state of the network and about the performance of PeerStreamer. We use Internet Control Message Protocol (ICMP) Echo messages to measure the reachability and Round Trip Time (RTT) between each couple of nodes over the IPv4 WCN. We also measure the ratio of successfully arrived video chunks and the delay from their generation at the source node.

We analyze the data collected in the five central minutes of each run, i.e., we discard the data at the beginning and end of each run to consider only the steady state behaviour of our system.

ICMP data is analysed to extract RTT mean and standard deviation in order to have an idea of the underlying network during the peer-to-peer overlay. The rate of ICMP packets is low enough not to interfere with the experiment. The code for the logs analysis is freely available[11].

In conditions similar to those of these experiments, i.e., a small overlay with nodes highly clustered, PeerStreamer has been shown to work perfectly (i.e., no chunk losses or excessive chunk delivery time) in NAPA-WINE deliverables, and in experiments in our campus WiFi network. We do not reports the relative results as they do not represent a real comparison, and can be mistakenly interpreted ad a proof that Community-Lab is not working correctly, while instead we think that Community-Lab results are representative of the real challenges faced into a WCN by a live streaming application, challenges obviously not present in a WiFi network just serving as access to a resource-rich campus network.

[11] http://halo.disi.unitn.it/baldesi/PublicGits/peerstreamer_logs_analyzer.git/

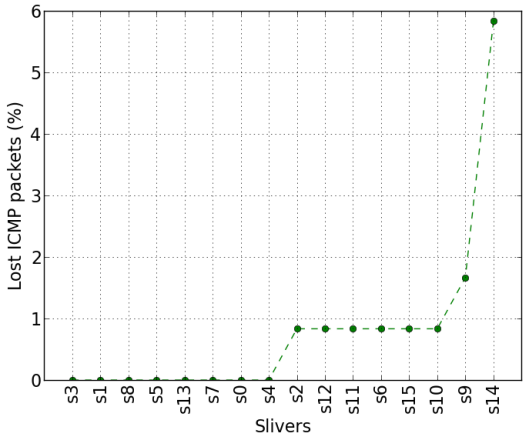

Fig. 3. ICMP traffic loss percentage with respect to the streaming source during the experiment for each of the involved slivers.

IV. Results and Discussion

The experiments collect measures and performance both at the IP level through ICMP messages and at the applications level directly within PeerStreamer. First in Sect. IV-A we discuss the basic IP level performance and characteristics, whose behavior is fundamental to understand the performance of the application above it, which is discussed in the remaining subsections.

A. The Community-Lab Connectivity

As said, we had to use only slivers in the Guifi.net network in Cataluña. Nevertheless, the Guifi.net network is itself a collection of separate islands that can be physically far from each other. We measured exactly this kind of behaviour using ICMP. We selected 10 slivers belonging to an island, and 6 to another one, and this is reflected by their network performance. Figs. 3 and 4 show packet loss and RTT measured from any sliver with respect to the one acting as the video source (the *source*, from now on). It is clear that some of the slivers have much better connectivity than the others and in particular the slivers s14 and s15 have a very poor condition toward the source and all the slivers in the other island.

Fig. 5 reports the average packet loss experienced by each sliver toward all other slivers in the test. Again, we see that s14 and s15 experience a network performance much worse than the rest of the slivers. Sliver s15 in particular is subject to a large number of out-of-scale (larger than 200 ms) RTT values toward the source.

For this reason the plots in Fig. 6 exclude s15, in order to have a smaller scale and an easier to read graph. Fig. 6

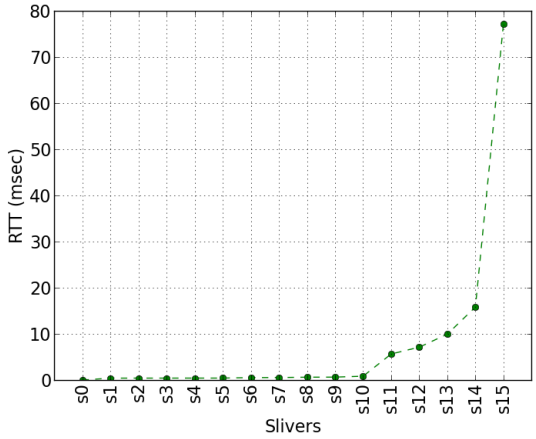

Fig. 4. Average RTTs of the ICMP traffic from all the slivers to the source.

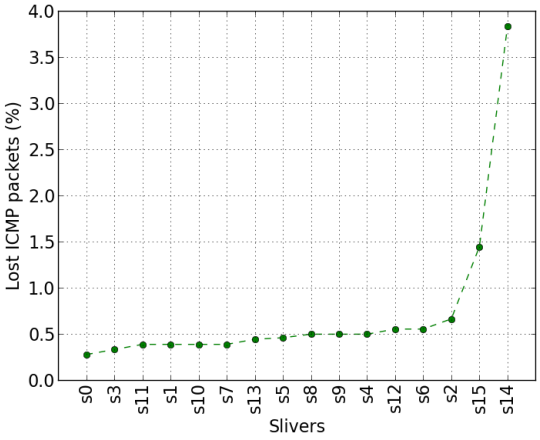

Fig. 5. ICMP traffic loss during the experiments for each involved slivers.

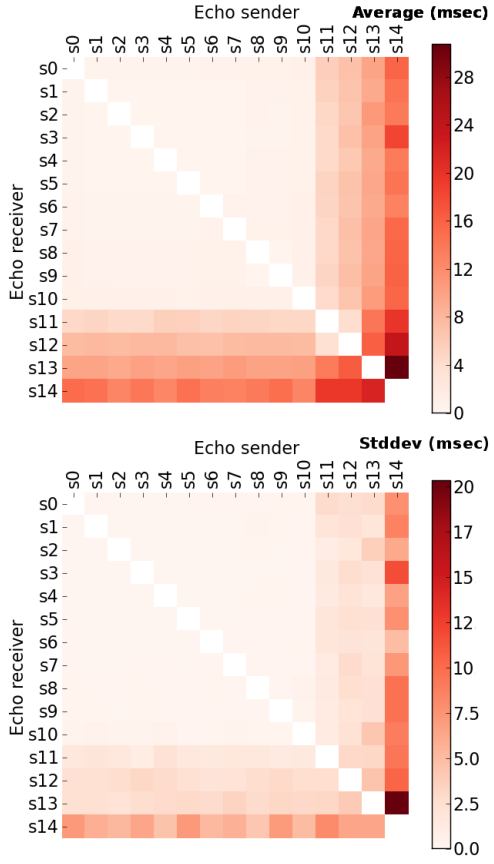

Fig. 6. Upper plot: average RTT from each sliver to the others, slivers are sorted based on their RTT with the source; bottom plot: RTT standard deviation from each sliver to the others.

shows two matrices representing the average and the standard deviation of the RTT value between every pair of slivers. These data confirm that in our experiments we used slivers coming from two different islands, that show different connectivity conditions. In the first island we have nodes with small RTT and little variation, in the second island the nodes have worse performances, larger RTT and higher variations.

These measures are very important to understand the behaviour of Community-Lab and PeerStreamer. For Community-Lab we can say that in its present state it offers an efficient platform to set-up, run and manage experiments, but still it has to mature to be really representative of WCNs. In fact, analysis of WCN topologies [16], [17] have shown that WCNs generally offer stable paths and performances and many more nodes and hosts than available in Community-Lab. The heterogeneous conditions we found in our experiments probably depend more on the development stage of Community-Lab than on the state of the underlying WCNs. Indeed, such conditions offer an interesting ground to test PeerStreamer. Given the features of the underlying network we expect PeerStreamer to rely principally on the nodes that show a better average connectivity and the next section will show that this effectively happens.

B. *PeerStreamer Overlay Topology*

In order to distribute the video chunks, PeerStreamer sets up an overlay network using complex metrics, partially described in (Sect. II-A). This overlay is time-variant to counter churn (not present in these experiments, but fundamental in driving P2P systems performance), but its average behaviour can be represented with an adjacency matrix showing the number of chunks exchanged from one peer to another. Fig. 7 shows the number of chunks sent between every pair of slivers during a specific run, thus giving a pictorial representation of the average behavior of the streaming session. Averaging these results over multiple runs is not meaningful; however, the other runs behavior is coherent with the one represented here.

The first thing to note is that the first column of Fig. 7 has a uniform color on all rows. PeerStreamer in fact uses a simple distribution strategy for the source: the original video chunk is sent to a peer chosen with uniform probability. This grants a reliable and robust delivery even in unstable scenarios in which peers might suddenly disconnect. The second thing to note is that the comparison between Fig. 7 and Fig. 6 shows that the majority of the chunks are sent from the slivers that show better connectivity. This confirms that PeerStreamer is able to exploit the RTT and link loss in order to obtain the best delivery network and concentrate the efforts on the more virtuous slivers. Finally, we see that each sliver receives a

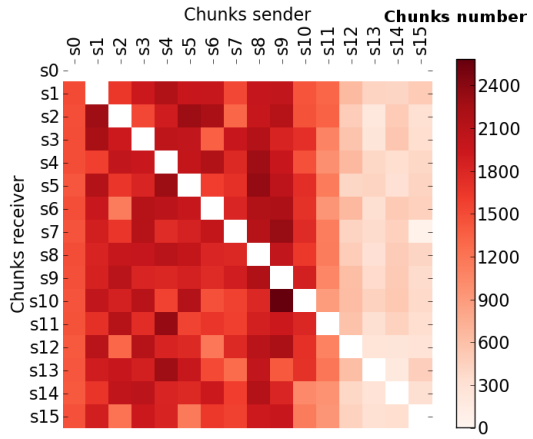

Fig. 7. Chunks exchanged during a run of the experiment; the color of each block represents the number of chunks exchanged from the sliver on the x-axis to the sliver on the y-axis.

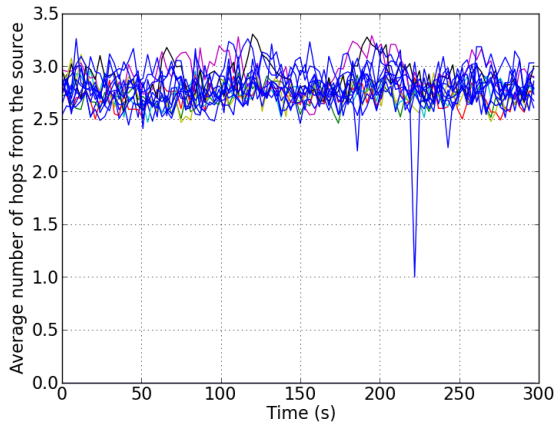

Fig. 8. Variation of hops of the delivered chunks per peer during a run.

fair amount of chunks from the other ones, and that no sliver behaves as a bottleneck for the whole network.

Fig. 8 shows the average number of hops from the source to all destinations versus time. In fact each received chunk has a field indicating the number of peers it visited from the source before the reception. Depending on the initial destination of the newly generated chunk, which is random according to Fig. 7, the hops number can change. Fig. 8 report the moving average taken every few seconds on every sliver. The average value is quite stable for each peer and approximately constant. Fig. 9 shows instead the distribution of the number of hops for all received chunks, for each peer. Again, we see that the distributions are all quite similar, which guarantees a fair distribution of the delays for the received chunks among all the slivers.

These results show that the algorithms that PeerStreamer uses to build its topology behave well also in the challenging scenario of the Community-Lab: PeerStreamer exploits the nodes that have more resources, but does not create bottlenecks, every sliver receives chunks with a similar distribution of hop count and the distribution graph is dynamically changed.

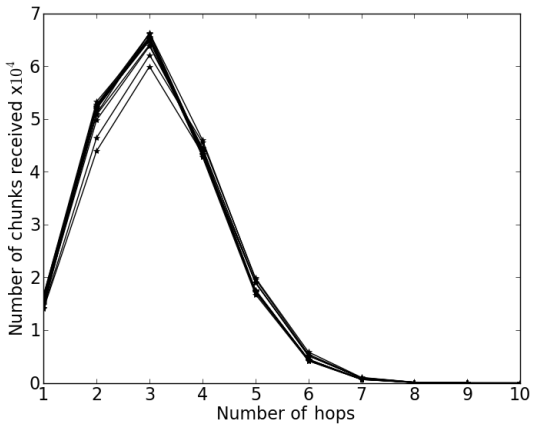

Fig. 9. Number of the overall chunks received by the peers versus the number of hops of the chunks.

As already remarked dynamism is important in peer-to-peer video streaming: not only churn may change the peers participating in the distribution, but also the network conditions may vary quickly and the distribution graph must be able to cope with them.

C. Chunk Loss and Accuracy

Fig. 10 shows the overall percentage of lost chunks during an experiment. We can easily note how the performance reflects the connectivity conditions measured with ICMP messages. The two slivers that present the highest loss, and also the highest RTT to the other slivers (see Fig. 6) are the ones that loose the highest number of chunks. Fig. 11 represents the fraction of received video chunks averaged for time windows of fixed length during the evolution of the experiment. The two graphs show the data for the first 14 slivers (the source is excluded) and the data for s15 separately. As said, s15 is the sliver that shows the worst performance in terms of RTT to any other sliver in the experiment. This heavily influences the number of lost chunks, even more than the loss rate. To understand this, consider that PeerStreamer is designed for live streaming so that receiving an old chunk is not useful. Since s15 has very high RTTs with the other slivers, it is not able to contribute to the overall distribution process, but it also receives the chunks with high delays. As a consequence, some of the chunks, even if available to be transferred, are simply considered uninteresting by s15. Fig. 11 shows that for the whole experiments the chunk loss is almost always lower than 10% and for the majority of the time lower than 5%. These are values that allow to say that video streaming is viable on the Community-Lab and that the presence of a peer with vary bad performances does not impact the performance of the rest of the peers. Notice that improving slightly the networking conditions, having the source injecting multiple copies of the chunks in the overlay, and developing a customized incarnation of PeerStreamer may quickly lead to a much better performance at the application level, even if the network conditions remains so harsh.

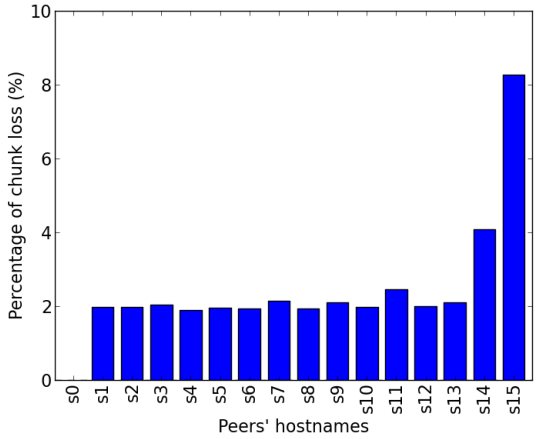

Fig. 10. Overall chunk loss during the experiment for each sliver.

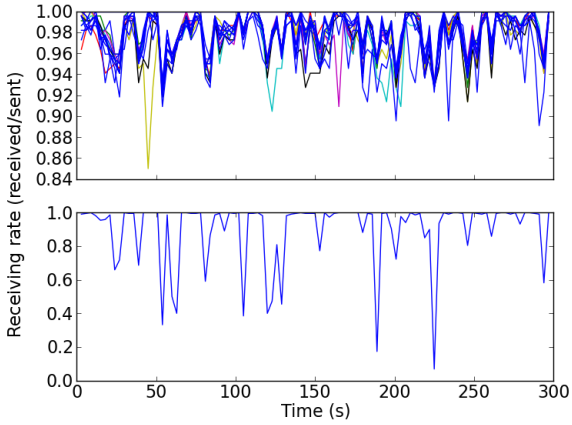

Fig. 11. Fraction of received chunks for each peer along one experiment. First graphs shows data for slivers s0-s14, second graph for sliver s15.

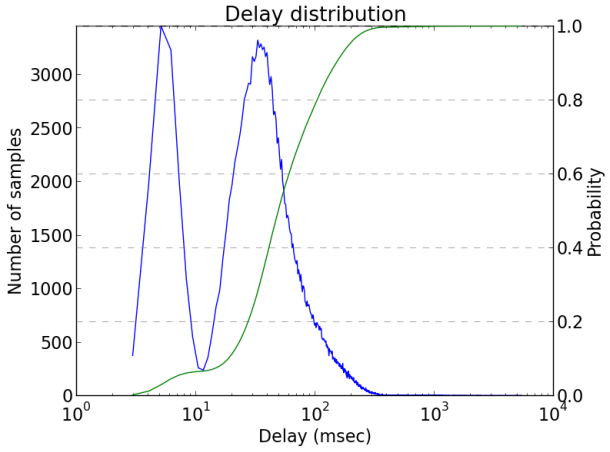

Fig. 12. Overall chunk delay distribution for sliver s12.

D. Chunk delay

Fig. 12 shows the chunk delay distribution during the experiment for one sliver. For the sake of clarity we chose to plot the data relative to one sliver only, and to use a log scale, otherwise the plot would be too much compressed. Note that using a log scale for the x-axis makes the area underlying the blue curve not representative of the cumulative number of samples, for this reason we also report the cumulative distribution as the green curve.

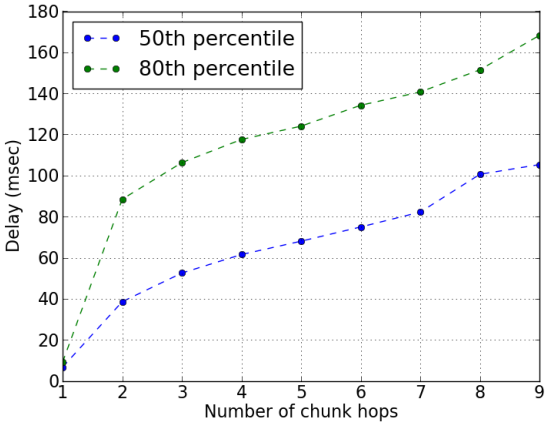

Fig. 13. Median chunk delay grouped by number of hops for sliver s12.

From the figure we can see that eighty percent of the packets arrive within 110 ms which is a very good delay for live streaming, and it would even be acceptable for a video call or video conferencing. Fig. 12 shows also the presence of two main modes in the distribution of delays, which can be explained looking at Fig. 13, which reports the 50-th and 80-th percentile of the delivery delay versus the number of transmission hops of the chunk. The chunks that are received directly from the source always present a very short delay (the 50-th and 80-th percentiles are very similar): these chunks are pushed by the source, so their delay is only half the RTT from the source plus obviously the transmission delay. The delay sharply increases for packets received after two hops, and also the 80-th percentile diverges more from the median (50-th percentile), and a then smoothly increases for paths longer than 2 hops. This is highly positive, since it shows that PeerStreamer can handle the distribution of chunks on long paths keeping the overall delay low, note however that the number of chunks delivered using more than 4 hops is limited (see Fig. 9), so that the overall delay distribution is more favourable, as seen in Fig. 12.

V. Conclusions

A. Community-Lab Conclusions and Best Practice

Since different WCNs in CONFINE are missing a direct interconnection, streaming across different WCN is not feasible, not even among islands of the Community-Lab, thus we had to limit experiments to Community-Lab slivers on a single WCN. This limitation can be overcome on the real WCN exploiting of external services, like the one provided by STUN servers [18]. The WCN nature of the testbed is a challenging scenario to deal with, but, since the growing audience around WCNs, it is also of major interest from the research the point of view. The initial Community-Lab network analysis, which gives results showed in Sect. IV-A, indicates the testbed well support even resource hungry applications.

During tests we developed our framework to work with Community-Lab APIs. That effort is an initial step in order

to perform further experiments both from our side and from other researchers side. The limitations like the presence of slivers not mutually reachable, the lack of system clock synchronization, and the limited number of deployed testbed nodes, are for the moment preventing a deeper analysis in the video streaming context. Nevertheless, Community-Lab is still a work in progress and a great chance to perform experiments *hands on* on real WCNs infrastructures and resources.

B. Live Streaming Conclusions

Live peer-to-peer video streaming using PeerStreamer within WCNs is already possible, even on the virtualized and limited environment offered by the Community-Lab. Results in Sect. IV-B compared with the findings of Sect. IV-A, illustrates that, the way used by PeerStreamer to build up the distribution overlay, exploits important network features like links RTT and loss. Sect. IV-C presents how the built topology and the underlying network led to a live content distribution with contained loss, which is one of the basic conditions to achieve a good quality of video streaming. From the results shown in Sect. IV we can then conclude the feasibility of live video streaming in the WCNs context and the reliability of the underlying Community-Lab.

C. Future Work

PeerStreamer has generally proven to well suite on large scale, Internet-like networks. A more specific tailoring for WCNs networks could improve performances. Especially during topology management, dealing with such occasionally lossy link as reported in Sect. IV-A, a dedicated peers selection algorithm during initial chunk injection in the overlay could bring to avoid an excessive initial chunk loss. That strategy obviously would decrease the overlay stability and thus should be further studied.

REFERENCES

[1] S. Jain and D. Agrawal, "Wireless community networks," *IEEE Computer*, vol. 36, no. 8, 2003.

[2] R. Battiti, R. Lo Cigno, M. Sabel, F. Orava, and B. Pehrson, "Wireless LANs: From WarChalking to Open Access Networks," *Mob. Netw. Appl.*, vol. 10, no. 3, pp. 275–287, Jun. 2005.

[3] J. Barceló, A. Sfairopoulou, and B. Bellalta, "Wireless Open Metropolitan Area Networks," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 12, no. 3, pp. 34–44, Jul. 2008.

[4] P. Frangoudis, G. Polyzos, and V. Kemerlis, "Wireless community networks: an alternative approach for nomadic broadband network access," *IEEE Communications Magazine*, vol. 49, no. 5, 2011.

[5] R. Lo Cigno, "Untethered Local Communications: From Wireless Access to Social Glue," in *IEEE Wireless On-demand Network Systems and Services (WONS 2011)*, Jan. 2011, pp. 42–43.

[6] M. Kas, S. Appala, C. Wang, K. Carley, L. Carley, and O. Tonguz, "What if wireless routers were social? approaching wireless mesh networks from a social networks perspective," *IEEE Wireless Communications*, vol. 19, no. 6, pp. 36–43, 2012.

[7] A. Alasaad, "Content sharing and distribution in wireless community networks," University of British Columbia; http:\\circle.ubc.ca/handle/2429/44205, PhD Thesis, April 2013.

[8] A. Neumann, I. Vilata, X. León, P. E. Garcia, L. Navarro, and E. López, "Community-lab: Architecture of a community networking testbed for the future internet," in *IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), Barcelona, Spain*, 2012.

[9] 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," *Communications Magazine, IEEE*, vol. 49, no. 6, 2011.

[10] L. Abeni, C. Kiraly, R. Russo, M. Biazzini, and R. Lo Cigno, "Design and Implementation of a Generic Library for P2P Streaming," in *In Workshop on Advanced Video Streaming Techniques for Peer-to-Peer Networks and Social Networking*, Florence, Italy, Oct. 2010.

[11] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia, "Experimental comparison of neighborhood filtering strategies in unstructured P2P-TV systems," in *Proc. of the IEEE Int. Conf. on Peer-to-Peer Computing (P2P'XII)*, Sept. 3-5, 2012, pp. 13–24.

[12] L. Abeni, C. Kiraly, and R. Lo Cigno, "On the Optimal Scheduling of Streaming Applications in Unstructured Meshes," in *In IFIP Networking*, Aachen, Germany, May 2009.

[13] A. Russo and R. Lo Cigno, "Delay-Aware Push/Pull Protocols for Live Video Streaming in P2P Systems," in *In IEEE International Conference on Communications (ICC'10)*, May 2010.

[14] R. Birke, C. Kiraly, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "Hose rate control for P2P-TV streaming systems," in *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, Aug 2011, pp. 202–205.

[15] L. Peterson, R. Ricci, A. Falk, and J. Chase, "Slice-based federation architecture," *Ad Hoc Design Document (July 2008)*, 2010.

[16] L. Maccari, "An analysis of the Ninux wireless community network," in *9th IEEE Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2013, pp. 1–7.

[17] D. Vega, L. Cerda-Alabern, L. Navarro, and R. Meseguer, "Topology patterns of a community network: Guifi.net," in *2012 IEEE 8th International Conference on Wireless and Mobile Computing, Networking and Communications WiMob*, 2012, pp. 612–619.

[18] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for nat (stun)," Internet Requests for Comments, 2008.