Improving P2P Streaming in Community-Lab Through Local Strategies

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*—Distributing live streaming in Wireless Community Networks (WCNs) is a service with a high added value; however, cloud-based streaming, as commonly used in the Internet, does not fit well the architecture of WCNs, which often have restricted access to the Internet. Modern WCNs, instead, can have a good internal connectivity with high bandwidth. A P2P approach is thus well matched for streaming in WCNs. This paper presents experimental results obtained with PeerStreamer running on top of Community-Lab, a test-bed realized by the CONFINE EU Project for the experimentation of novel protocols in community networks. The experiments highlight relevant differences between a WCN and the Internet, and we propose strategies that can be implemented on all the peers or even only locally on the source to improve the streaming quality. These strategies are based on simple heuristics and can be activated dynamically when the streaming quality degrades below a given threshold.**

I. Introduction and Background

Wireless Community Networks (WCNs) are a lively instrument to foster a bottom-up approach to broadband networking with a clear slant toward innovative and socially sustainable communication models. Since they are growing, the demand for advanced community services is also increasing. Streaming of live events *within* the WCN without the need of Internet connection is one of these services. Regional or city events can be broadcasted to the users, and a single proxy on the periphery can serve Internet-based events to the entire WCN using a minimal amount of external resources.

Streaming in WCNs must be efficient and properly mapped to the WCN structure and characteristics. Key factors for successful live streaming are low delays and efficient resource use. WCNs, however, rarely have large data-centers with the computing and networking resources needed to stream contents to hundreds of users at the same time (acceptable video quality can be obtained with 300 kbit/s, and standard TV quality may require as much as 2-3 Mbit/s). On the other hand, they are evolving toward a very good availability of bandwidth (tens to hundreds of Mbit/s) evenly distributed within the WCN.

A P2P approach seems thus a perfect match for this challenge, mapping very well to both the technical features of WCNs and their societal goals. However, ensuring a timely delivery with a P2P approach, minimizing the needed resources and adapting to the network characteristic is still a major challenge. "Standard" P2P-TV services like PPLive, UUSee, PPStream or SopCast[1] are not open source, and thus they do not allow the needed level of customization, nor are they really a good match for the WCN networking model.

This paper tackles the problem of adapting and customizing a P2P streaming application in WCNs by experimenting in the Community-Lab facility provided by the CONFINE project. The selected P2P platform is PeerStreamer, that we have developed during NAPA-WINE[2], and we are maintaining as an open source project. Since WCNs infrastructure differs a lot from the Internet in general, the results we obtained in [1], can only be used as guidelines and are not direcly applicable to WCNs for which the characteristics must be studied.

The contribution of this paper is twofold. On the one hand, we extend and complete the analysis of Community-Lab functionality that we have partially presented in [2]; on the other hand, we present the first set of scientific experiments exploring the possibility of distributing live video streams on a WCN exploiting its intrinsic networking properties.

II. P2P Streaming with PeerStreamer

In a peer-to-peer distribution, the source "seeds" the overlay with one or more copies of the content to be distributed, and peers contribute by exchanging the content among themselves. We use a mesh-based overlay (without a structured topology) and a single live video stream. The content is generated by the source "on the fly", as in live recording, by periodically emitting video and audio frames, which are separately aggregated into chunks for the peer-to-peer dissemination. We stress on the *live* characteristics of the streaming, that is crucial for applications like video conferencing, so the delivery of the frames must be time-bounded. Video and audio frames are independently generated and they are distributed in different "chunks", but on the same overlay.

The distributed nature of the dissemination and the cooperative assumption of the P2P paradigm adapts well to the network infrastructure of a WCN. The WCN principles of network neutrality and resource sharing naturally also match perfectly with P2P philosophy and strategies.

PeerStreamer is a P2P live streaming platform capable of disseminating real time content among thousands of nodes, and structured around the concept of both content- and network-awareness [3]. It provides the application developer with

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"

[1] http://www.pplive.com, http://www.uusee.com, http://www.ppstream.com, http://www.sopcast.com

[2] http://napa-wine.eu, http://www.peerstreamer.org

978-1-4799-5041-6/14/$31.00 ©2014 IEEE

a set of open source, efficient core libraries [4], as well as more advanced routines and algorithms for the efficient scheduling of chunks [5]–[7], and topology management [1]. PeerStreamer guarantees that, even dealing with overlays of thousand of nodes, the content delivery delay remains in the order of seconds and it is almost immune to churn, thanks to the extremely dynamic topology management. It is easily customizable in different ways, but previous results suggest to avoid constant chunk size configurations [8]. In the rest of the section we will explain three concepts at the base of PeerStreamer: the way the overlay is built, the policy for chunk dissemination and the selection of peers and chunks for scheduling content distribution.

A. Overlay of Peers

Suppose to install and launch PeerStreamer on several nodes for a time period of length T. Let $S(t)$ be the set of all peers (PeerStreamer instances) at time $t \in [0, T]$. $S = \{\text{peer } P : \exists\, t \in [0, T] \text{ such that } P \in S(t)\}$, is the set of all peers and P_i the i-th peer of S (we use the subscript i only to distinguish one peer from the other, the way peers are ordered is not relevant).

PeerStreamer maintains the overlay topology, and continuously adapts it during the evolution of the stream, with a two steps approach. First, a *peer sampler* module uses a modified version of Newscast [9], [10] (a gossiping algorithm) to take random samples of the peers in $S(t)$. As a result, each peer knows at least a subset of $S(t)$ at every t. Next, a topology management module filters the sequence of these samples to select a subset of size NN that form the neighborhood of the peer. Thus, PeerStreamer topology is described by a directed graph and the topology management controls the outgoing links forcing an NN-regular topology (a topology with constant degree NN) for the outgoing links. The incoming links are also selected to try matching the outgoing ones and favor symmetric relationships, so that also the incoming topology is roughly NN. To guarantee good connectivity of the overlay $NN > \log_2(||S||)$; depending on the underlying network characteristics, a much larger NN can help the dissemination process, and in general if $||S||$ is small, then NN should be relatively larger.

Let $N_i(t)$ be the set of all peers in the neighbourhood of P_i at time t.

Definition The peers overlay built by the topology manager during the experiment is a directed graph:

$$R_N(t) = (S, L_N(t))$$

where S is the set of nodes and

$$L_N(t) = \{(P_i, P_j) : P_j \in N_i(t)\}$$

is the set of links.

Since $N_i(t)$ is constantly changing, also $R_N(t)$ is constantly changing.

B. Chunk Dissemination

In live streaming the source is a special peer $P_s \in S$ that provides the media content and must always be part of $R_N(t)$. In PeerStreamer, P_s concatenates one or more frames (audio and video separately) into chunks and injects them in the overlay $R_N(t)$. Chunks are numbered; C is the set of all chunks generated by P_s and C_h is the h-th chunk. For each chunk C_h there is a delivery deadline dl_h related to the playout time, after which C_h is no more useful and it's "lost" for the application.

After its generation, a chunk C_h is injected in $R_N(t)$: P_s selects a peer $P_i \in N_s(t)$ and pushes C_h to it. C_h can be injected in m multiple copies to different peers in $N_s(t)$ in order to speed up the dissemination process. Chunk reception is always acked (by all peers) for reliability and to avoid duplications. At regular time intervals τ_o, every P_i "offers" a window of size n_w of its most recent chunks to one $P_j \in N_i(t)$. P_j "selects" a set of size sc of them so that every C_h percolates in $R_N(t)$ using an *offer/select* protocol.

C. Peer and Chunk Selection

We refer to *peer and chunk selection* as the procedure used by P_i to choose $P_j \in N_i(t)$, offer it a set of chunks and let P_j selects the sc of them he needs more.

Many works have been published studying strategies to disseminate chunks in a P2P overlay in the Internet [6], [7], [11], [12]; some of them were never proven better than a random-random strategies in realistic conditions (i.e., select one peer and one chunk at random). PeerStreamer has been designed to work on the Internet and we consider the "benchmark" scheduler one that selects for the offer $P_j \in N_i(t)$ uniformly at random, while P_j selects the most recent chunk that it does still not possess. This is called a "random – latest-useful" (R-LU) chunk selection and with some assumptions it can be considered optimal for live streaming [5].

However, WCNs are not the Internet and some special adaptation might be needed, i.e., R-LU continues to be a benchmark, but adaptation procedures, e.g., based on the Round Trip Time (RTT) measures, might not work, as RTT in a WCNs is not stable and dominated by propagation, but can be highly variable and random, as we experienced in past experiments [2]. Moreover in WCNs, $||S||$ can be expected to be relatively small (tens to hundreds of users, not thousands or millions), and this can also affect the adaptation strategies, making "wide Internet" approaches suboptimal.

The experiments presented in this paper regards the behavior of the distribution as a function of five different parameters:

- NN: the target dimension of the outgoing neighborhood;
- m: the number of copies of each chunk injected by P_s into the overlay;
- sc: the number of chunks that a peer is allowed to select from each offer it receives (specified in the offer);
- fa: the number of audio frames that are assembled into

a single audio chunk[3].

- P_s seeding strategy: Au or Aw. When the source P_s generates a new chunk it must decide to which peer $P_i \in N_s(t)$ it will be sent. The choice can be random with a uniform distribution (Au), or it can follow a weighted distribution that takes into account the local communication quality between P_s and P_i with the goal of making a more reliable chunk injection in the overlay $R_N(t)$ (Aw).
  Let $w_i(t)$ be the weight associated by P_s to $P_i \in N_s(t)$ at time t. When a previously unknown peer P_i enters $N_s(t)$ its weight $w(i)$ is set to 1. P_s selects $P_i \in N_s(t)$ at time t with probability

$$p_i^s(t) = \frac{w_i(t)}{\sum_{j:P_j \in N_s(t)} w_j(t)} \tag{1}$$

  If $\forall\ t > 0$, $w_i(t) = 1$ then $P_i \in N_s(t)$ will be selected with uniform probability, according to Au. To implement Aw we choose as weight for each peer the rate of chunks correctly acknowledged, computed by P_s (Eq. 2), as a passive a posteriori measurement of links quality. For each $P_i \in N_s(t)$, P_s computes $w_i(t)$ as a moving average of the successful chunk delivery rate.
  Once P_i has been selected by P_s for chunk injection, P_s sends the chunk and updates its weight:

$$w_i(t) = \begin{cases} \alpha + w_i(t)(1-\alpha) & \text{if ack received by } P_s \\ w_i(t)(1-\alpha) & \text{if a timeout expires} \end{cases} \tag{2}$$

  Currently we have set $\alpha = 0.01$ and a timeout of 10 ms. The benefits of this strategies are the flexibility, since it does not depend on any other component except the chunk dissemination module, the relevancy of the weights, since they are evaluated on the chunk loss itself and the absence of further data transmission on the overlay.

III. Experimental Setup

In order to have a reasonably controlled environment, and the possibility to run experiments through a standard and centralized interface, we run the experiments on the facilities provided by the CONFINE EU project: the Community-Lab.

A. Community-Lab

Community-Lab [13] is a testbed linking together nodes from different WCNs, intended for the investigation of WCN issues and solutions in realistic scenarios. Currently the CONFINE project involves different communities spread across Europe: Guifi.net, AWMN, FunkFeuer and Ninux.org, placed respectively in Spain, Greece, Austria and Italy.

Community-Lab provides an easy interface for driving experiments over WCNs, allowing researchers to allocate portions of resources of the network, take control of virtual machines placed inside a WCN and have a global view of the currently available resources.

Community-Lab is made up of special nodes called Research Devices, which are directly connected to WCN nodes, so they actually communicate one another through the WCN. Each research device can instantiate multiple virtual machines called slivers, each of which belongs to a group called a slice. Researchers can create a slice and the related slivers by selecting the desired research devices and then launch experiments on them with a single command. Those functionalities are achieved through a testbed server which is accessible by researchers through a management VPN.

B. Experiments Management

Since the Community-Lab testbed server provides a standard ReST interface to manage slices, we realized a `bash` framework for driving the experiments. The code is released as open source on the web[4]. Once a researcher has registered on the Community-Lab web interface, created a slice, launched the slivers and added his personal computer to the management VPN, it can use this framework to drive experiments, as we have documented in [2]. Thus, our results are perfectly reproducible for any researchers with access to the Community-Lab.

C. Experiments issues and workaround

As noted in [2], slivers are not always time synchronized and due to the virtualization it's not possible for the researchers to manipulate the system clock. Our experiments are influenced by timing and we are especially interested in evaluating the chunks' delay. During the test runs we log the data together with the system timestamps in order to draw conclusions over timing and network evolution. To trace the time difference among timestamps of different machines in the same instant, we periodically perform an NTP query from each peer of the overlay and the source (taken as time reference) and log the result. During data preprocessing we perform a timestamp rescaling of peers data by interpolating the time difference obtained with the NTP queries from each peer with respect to the source.

Sometimes, experimenting in the Guifi.net island, we experienced some vary bad communication conditions. In order to trace those unlikely scenarios, which could compromise the meaningfulness of the experiments, we periodically generate from each peer to every other peer in the overlay some Internet Control Message Protocol (ICMP) traffic. When the ICMP logs show that there are peers experiencing an ICMP data loss greater than 30%, we assume that the system is not working correctly and results are not meaningful; we immediately stop the experiment and discard it entirely. Note that we cannot control the actual load of Research Devices, since multiple experiments may be run independently by several researchers. A high loss of ICMP packets can be a symptom that the machine hosting the Research Device is simply overloaded and does not respond in time, rather than a wireless link

[3]The video is H.264 encoded; the encoder generates video frames of variable dimension and audio frames of fixed size (207 bytes), which are output separately and without a strict timing relation. A fixed ratio of 1 video frame per chunk maintains a very low chunkization delay, but we can explore how much gain can be achieved by assembling together fa audio frames thus reducing the offer/select traffic overhead

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

of the WCN is experiencing a sudden crisis. We consider 30% loss rate for small packets as pathological and not a behavior expected in standard 802.11 links[5]. Our scripts for data preprocessing are freely available on the web[6].

D. Experiments Performed

To gain as much insight into the problem as we can, we performed our tests in two islands of Community-Lab: Guifi.net and AWMN. Since Community-Lab is still an ongoing project, the number of slivers in the other islands is still too small to be representative for our experiments, and also it is not yet possible to interconnect different Community-Lab islands.

Every experiment is composed of several runs (normally between 10 and 30), each of which lasts ten minutes; we analyse the central five minutes of the runs in order to avoid data related to transient behaviors. The data of the experiments is averaged over all the successful runs.

The video we distribute is a re-encoding at 24 fps (video frames) and average bit rate of 300 kbit/s (including both audio and video) of Big Buck Bunny[7]. τ_o is set to guarantee that on average the offer rate is slightly larger than the frame rate, so the distribution is sustainable and the signalling overhead is minimal: a much smaller τ_o easily guarantees a better distribution but at the price of many refused offers, increasing the overhead.

The experiments explore the influence of the five different adaptation strategies discussed in Sect. II-C. NN is varied from 5 (a value dangerously small) to 20 which, with the number of nodes we can deploy, means nearly a full mesh. Since $||S||$ is small we explore $m \in \{1, 3\}$ only; as m increase, the dissemination speeds up but it requires more and more resources at the node hosting P_s; as $m \to |S|$, the distribution process degenerates to a "multiple unicast" scenario as in cloud-based streaming. P_s can emit chunks either uniformly toward any peer (Au) or following Eq.2 (Aw). Finally, the number of chunks that can be selected for each offer is increased, $sc \in \{1, 3, 5\}$, and the number of audio frames per chunk is also increased, $fa \in \{1, 5\}$.

IV. Experimental Results

The measurement campaign we present consist of about one month of experiments, with roughly 100 hours of actual video distribution and several Gbytes of logged data analysed to select the most meaningful results. Experiments presented refer to the Guifi.net and AWMN Community-Lab islands. The number of available slivers is 10-12 in AWMN and 24-28 in Guifi.net, and we try to guarantee that for each experiment type the number of sliver is constant across different runs.

A. Underlying Network Performance

As mentioned in Sect. III-C, during tests we continuously monitor the state of the network through ICMP traffic. The frequency and the ICMP packet size are kept small in order not to interfere with the running experiment. During our experimentation on Guifi.net we collected the ICMP loss logs reported in Fig. 1 (upper plot). This test lasted seven hours and its results have been confirmed by several further experiments. The name of the slivers are unique identifiers in the Community-Lab. It's worth noting that slivers availability is not always granted, so different experiments can have a different number of slivers.

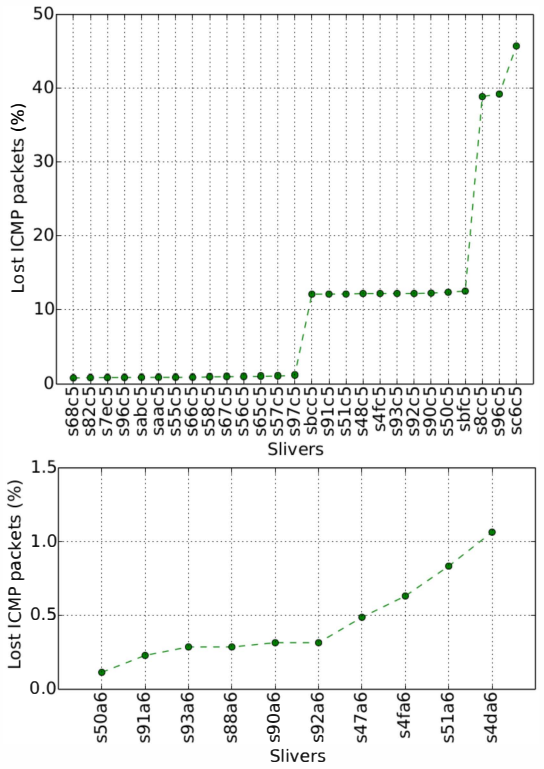

Fig. 1. Upper plot: ICMP loss on the Guifi.net island. Bottom plot: ICMP loss on the AWMN island.

In the Guifi island three groups of slivers are clearly identifiable; there is a major group of well connected slivers whose packets arrive almost always, another group of slivers experience bad communications losing more than 10% and the remaining three slivers are badly connected and suffer high loss of packets (around 40%).

During a similar seven hour experimentation on AWMN we collected the ICMP data reported in Fig. 1 (bottom plot). This network has a very little packet loss which is bounded by 1.2%.

B. Neighbourhood Size

First of all we analyze if NN plays a major role in performance. The test in Guifi.net has 24 slivers, while the one in AWMN 11. The results are shown in Fig. 2 and Fig. 3. Since the overlay over AWMN is composed of 11 peers only, the performance for NN=10 and NN=20 is almost identical, as the overlay is always a full mesh.

These results indicate that for these small overlays NN has a much higher impact than what was measured in [1].

[5]Recall that 802.11 implements MAC-layer explicit frame acknowledgement, so that an ICMP packet is really lost only when seven consecutive copies of it are lost on the channel.

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

[7]http://www.bigbuckbunny.org/

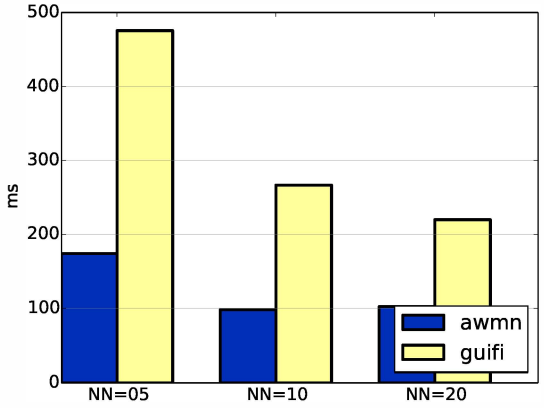

Fig. 2. Average chunk delay in the two WCNs, varying NN and $m = 1$.

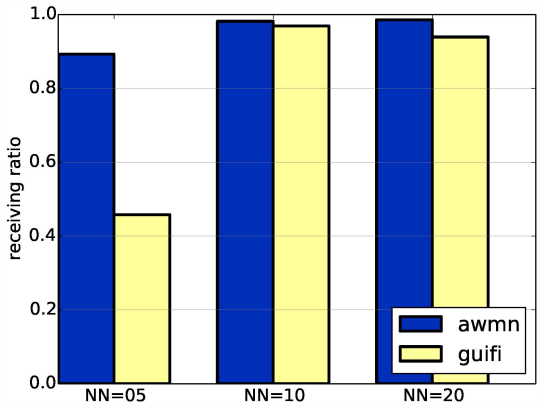

Fig. 3. Average chunk receiving ratio in the two WCNs, varying NN $m = 1$.

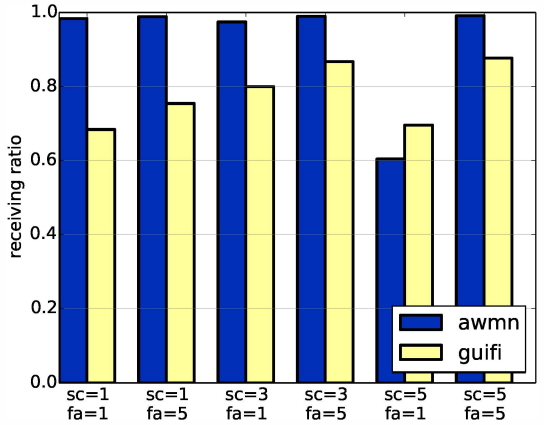

Fig. 4. Average chunk receiving ratio in the two WCNs, varying sc and fa and fixed $NN = 10$, $m = 1$.

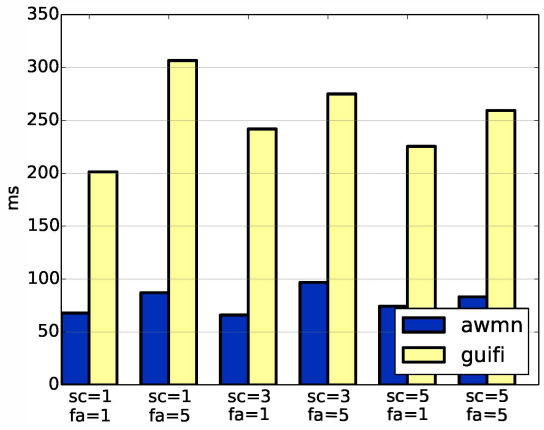

Fig. 5. Average chunk delay in the two WCNs, varying sc and fa and fixed $NN = 10$, $m = 1$.

A proper explanation requires further experiments with larger overlays in a WCN. In particular both Fig. 2 and Fig. 3 show that in Guifi.net NN=5 is definitely too small and performance is unacceptable, even if $NN > \log_2(||S||)$. NN=10 seems instead good for both networks and, if not otherwise stated the other experiments are run with this neighborhood size.

C. Chunks Transactions

Next we present the effects of chunk transaction dynamics, varying sc and fa in the offer/select protocol. The results are shown in Fig. 4 and Fig. 5. The names under the bars describe the relative configuration.

As can easily be observed in Fig. 4, a larger fa improves the receiving ratio. This is due to the consequent reduction of the total number of chunks per second, which requires a lower number of messages to be exchanged. Lowering the exchanged data overhead and the related reduction of message loss probability improve the overall data dissemination. As expected and illustrated in Fig. 5, a larger fa increases the delivery delay, but the effect is tolerable and remains within the limits of a live service. This effect is a clear consequence of the buffering of multiple audio frames, which should be delivered at different times, in a chunk sent when the last frame is ready.

The impact of sc on chunk losses is a bit more complex. Fig. 4 shows that delivery rate improves with sc for guifi.net, while in AWMN it is almost insensitive to sc, but for a degrade we cannot explain for the combination $sc = 5$, $fa = 1$. which indeed increases the chances that a peer can retrieve all the chunks he needs in due time. The impact of sc on delay (Fig. 5) is instead negligible.

D. Chunks Injection Multiplicity

As explained in Sect. II-B the source can inject multiple copies of each chunk in the overlay, helping a fast dissemination. This method requires that P_s has more networking resources, but it can greatly help the distribution process, specially in cases when the connectivity toward certain peers can have hard-to-predict outage periods. Our experiment involved 27 slivers in Guifi.net and 10 in AWMN; NN=10, sc=3, and fa=5. The results are illustrated in Fig. 6 and Fig. 7, reporting the average chunk delay and the average number of hops (in the overlay) per chunk respectively. The fraction of received chunks were almost constant and close to one, so they are not reported for the sake of brevity.

Fig. 7 shows that increasing m from 1 to 3 the number of average hops needed to disseminate the chunks from the source to the peers decreases. The result is expected, and it

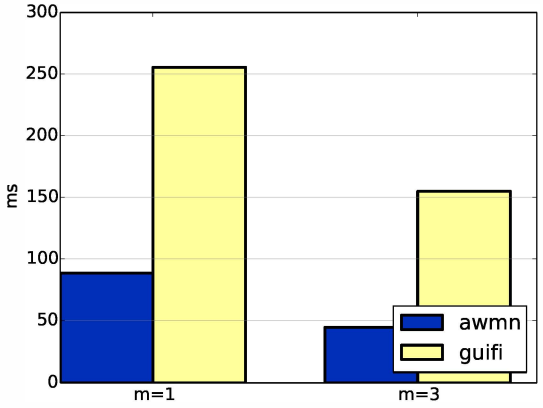

Fig. 6. Average chunk delay in the two WCNs, varying m; NN=10, sc=3, and fa=5.

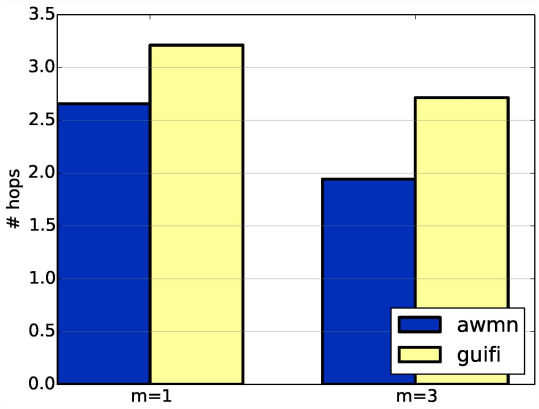

Fig. 7. Average number of chunk hops in the two WCNs, varying m; NN=10, sc=3, and fa=5.

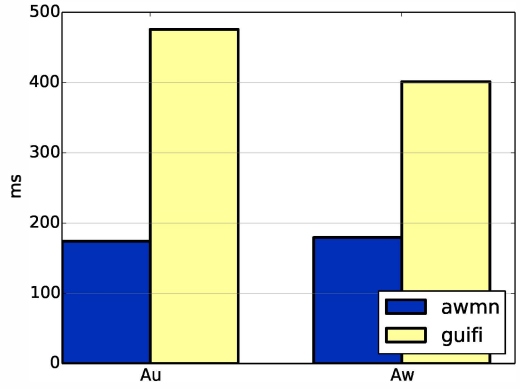

Fig. 8. Average chunk delay in the two WCNs, using the Au and Aw strategies, $NN = 5, m = 1$.

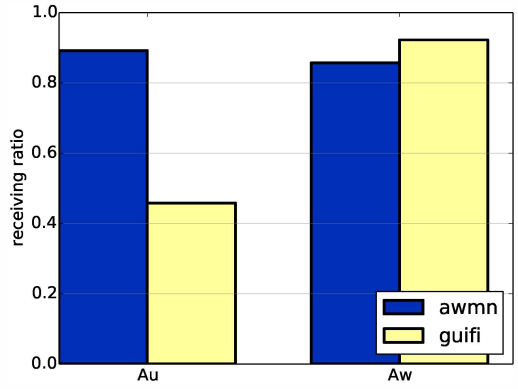

Fig. 9. Average chunk receiving ratio in the two WCNs, using the Au and Aw strategies, $NN = 5, m = 1$.

is a consequence of the fact that three peers have the newly created chunk at the same time, which roughly corresponds to have an overlay of size one-third for what the overlay graph diameter is concerned. As shown in Fig. 6, also the delay decreases remarkably in relation to the smaller number of dissemination steps required. We remark that these gains will not only remain, but they will be more evident in networks with hundreds of peers, where the dissemination process can be less uniform simply for stochastic reasons.

E. Push Strategy

Injecting multiple copies of the chunks at the source is effective, but requires more resources at P_s. The different injecting strategy devised in Sect. II-C can instead improve performance without requiring additional resources. The rationale is trying to select the peers that have the best connectivity to P_s, so that the probability that a chunk is lost at its first transmission (thus lost for ever and for everyone!) is minimized; if it lost in subsequent hops, some copies of it remains in the overlay for the dissemination. This reasoning can also be applied injecting multiple copies and replicating the strategy at every node, but we leave this analysis for future work.

The tests related to this solution involve 24 slivers in Guifi.net and 11 in AWMN; as in the previous results sc=3, and fa=5. We set NN=5 to enhance the performance difference, which however remains also for larger NNs.

The performance results are shown in Fig. 8 and Fig. 9. Since the AWMN networks characteristics are quite good, the Aw strategy does not offer relevant gains, as all the weights in (2) are roughly one. On the other hand, for Guifi.net both Fig. 8 and Fig. 9 show a notable improvement with respect to both delivery delay and receiving ratio.

It is interesting to notice, as Fig. 10 shows, that the average number of hops from the source to each peer in the overlay, increases with the Aw strategy. The reason is that chunks are injected less uniformly in the overlay, so that, on average, they have to be redistributed more times to reach all the peers, but this effect does not influence the delivery delay, as better transmission conditions imply that the chunks diffuse more evenly without the need of retransmissions due to chunks selected but never received correctly.

To gain more insight in this behavior, Fig. 11 reports the distribution of the average chunk delivery delay in Guifi.net, which has the same shape, and is almost not distinguishable in both Au and Aw strategies. So, how can the average be so different? The reason lies in the distribution tails, where "outliers" chunks can be diffused in several seconds increasing the average delay.

Summarising the results presented, it is not difficult to

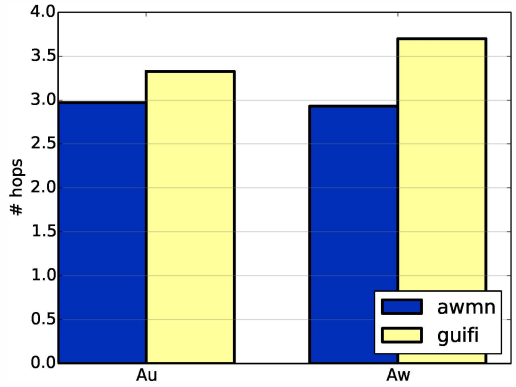

Fig. 10. Average numbers of hops from source to each peer, using the opportunistic and the random strategies, $NN = 5, m = 1$.

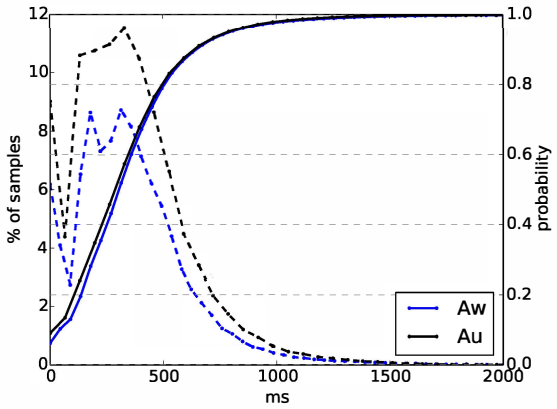

Fig. 11. Chunks delay distribution (dashed lines) and cumulative density function (continuous lines) during the tests on the Guifi island using the source random selection strategy (black lines) or the opportunistic selection strategy (blue lines). Data refers only to chunks arriving within two seconds.

identify a set of parameters and configurations of PeerStreamer that allow satisfactory live video distribution in WCNs, even when networking conditions are very tough as in the case of the Community-Lab Guifi.net island. Moreover, we can easily conceive automatic adaptation strategies, whereby, if the resource allow it, P_s switches to the emission of more copies per chunk $m > 1$, if feedback from the peers (e.g., collected and aggregated using the same gossiping protocol used to sample the overlay topology) indicates that the average performance is degrading below a certain threshold.

V. Discussion and Conclusions

The experiments presented in this paper are the first scientific evidence that a live, P2P video streaming distribution can be achieved in WCNs. We have adapted the PeerStreamer platform to run on Community-Lab, and, after solving issues related to the specific environment of Community-Lab, we have explored part of the parameter space that can be used to match a P2P, mesh-based live video streaming to the specific characteristics of a WCN.

The first observation is the complexity of providing a realistic and reproducible environment for WCNs experiments. Community-Lab is an extremely useful tool and provides an user-friendly environment; however, the number of slivers is still very limited, and tools are still missing to check the Research Device resources status during experiments, as well as to access information related to the WCN underneath, which means that applications like PeerStreamer which were designed to be network-aware and to adapt to networking conditions, are limited in their behavior by the lack of appropriate information.

The second and conclusive observation is the success of the experimental campaign, which provides very useful, albeit still initial, insight on the P2P video distribution performance achievable in WCNs. We have successfully shown that if the networking conditions are reasonably good, as in the Community-Lab island in AWMN, then the streaming achieves optimal quality without the need of any adaptation or tricks. If instead the networking conditions are very harsh, as in the Community-Lab island of Guifi.net, PeerStreamer need some tuning to achieve an acceptable quality. The good news is that in any case adaptations are not difficult, and can even be implemented as on-line autonomous behavior modification based on averaged feedback by peers. The feedback can be distributed and averaged on-line in the overlay exploiting the same gossiping protocol that is used for peers discovery.

References

[1] S. Traverso, L. Abeni, R. Birke, C. Kiraly, E. Leonardi, R. Lo Cigno, and M. Mellia, "Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison," *IEEE/ACM Trans. on Networking*, vol. 99, on-line, pp. 1–14, March 13, 2014.

[2] L. Baldesi, L. Maccari, and R. Lo Cigno, "Live P2P Streaming in CommunityLab: Experience and Insights," in *13th IEEE/IFIP Annual Mediterranean Ad Hoc Networking Workshop*, Piran, SLO, June 2014.

[3] 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 Comm. Mag.*, vol. 49, no. 6, 2011.

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

[5] 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.

[6] S. Traverso, C. Kiraly, E. Leonardi, and M. Mellia, "A performance comparison of hose rate controller approaches for P2P-TV applications," *Computer Networks*, vol. 69, pp. 101–120, 2014.

[7] A. Russo and R. Lo Cigno, "Delay-Aware Push/Pull Protocols for Live Video Streaming in P2P Systems," in *IEEE Int. Conf. on Communications (ICC'10)*, Cape Town, South Africa, May 2010.

[8] L. Abeni, C. Kiraly, and R. Lo Cigno, "Effects of P2P Streaming on Video Quality," in *IEEE Int. Conf. on Communications (ICC'10)*, Cape Town, South Africa, May 2010.

[9] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Proc. of Euro-Par'09*. Springer-Verlag, 2009, pp. 523–534.

[10] M. Barchetti and C. Kiraly, "Temporal correlation of Gossiping-based peer sampling methods," in *IEEE 13-th Int, Conf. on Peer-to-Peer Computing (P2P)*, Trento, Italy, Sept. 2013.

[11] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs." in *ACM SIGMETRICS*, Annapolis, Maryland, USA, June 2008, pp. 325–336.

[12] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *15th ACM Int. Conf. on Multimedia*, Augsburg, Germany, Sept. 2007.

[13] 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 Int. Conf. on Wireless and Mobile Computing, Networking and Communications (WiMob), Barcelona, Spain*, Oct. 2012.