POLITECNICO DI TORINO
Repository ISTITUZIONALE

Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison

Original
Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison / Traverso, Stefano; Luca, Abeni; Robert, Birke; Csaba, Kiraly; Leonardi, Emilio; Renato Lo, Cigno; Mellia, Marco. - In: IEEE-ACM TRANSACTIONS ON NETWORKING. - ISSN 1063-6692. - STAMPA. - 99(2015), pp. 741-754. [10.1109/TNET.2014.2307157]

Availability:
This version is available at: 11583/2558337 since: 2015-09-23T09:20:23Z

Publisher:
IEEE - INST ELECTRICAL ELECTRONICS ENGINEERS INC

Published
DOI:10.1109/TNET.2014.2307157

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

12 October 2024

Neighborhood Filtering Strategies for Overlay Construction in P2P-TV Systems: Design and Experimental Comparison

S. Traverso[a], L. Abeni[b], R. Birke[c], C. Kiraly[d], E. Leonardi[a], R. Lo Cigno[b], M. Mellia[a]
[a] DET, Politecnico di Torino, Italy – {lastname}@tlc.polito.it
[b] DISI, Univesity of Trento, Italy – {lastname}@disi.unitn.it
[c] IBM Research Lab. Zurich, CH – bir@zurich.ibm.com
[d] Bruno Kessler Foundation, Trento, Italy – {lastname}@fbk.eu

***Abstract*—Peer-to-Peer live-streaming (P2P-TV) systems aim at disseminating real time video content using Peer-to-Peer technology. Their performance is driven by the overlay topology, i.e., the virtual topology that peers use to exchange video chunks. Several proposals have been made in the past to optimize it, yet few experimental studies have corroborated results. The aim of this paper is to provide a comprehensive experimental comparison based on PeerStreamer, in order to benchmark different strategies for the construction and maintenance of the overlay topology in P2P-TV systems. We present only experimental results in which fully-distributed strategies are evaluated in both controlled experiments, and in the Internet, using thousands of peers.**

Results confirm that the topological properties of the overlay have a deep impact on both user quality of experience and network load. Strategies based solely on random peer selection are greatly outperformed by smart, yet simple and actually implementable strategies. The most performing strategy we devise guarantees to deliver almost all chunks to all peers with a play-out delay as low as 6 seconds even when system load approaches 1, and in almost adversarial network scenarios. PeerStreamer is Open Source to make results reproducible and allow further research by the community.

I. Introduction

In last years, Peer-to-Peer live streaming (P2P-TV) applications have emerged as valid alternative to offer cheap live video streaming over the Internet. Real-time TV programs are delivered to millions of users thanks to the high scalability and low costs of the P2P paradigm. Yet, exploiting the available resources at peers is a critical aspect, and calls for a very careful design and engineering of the application.

Similarly to file sharing P2P systems, in P2P-TV systems the video content is sliced in pieces called chunks, which are distributed onto an overlay topology (typically a generic mesh). But, contrary to file sharing P2P systems, chunks are generated in real time, sequentially and (in general) periodically. They must also be received by the peers within a *deadline* to enable real time playback. Chunk timely delivery is thus the key aspect of P2P-TV systems. This makes the designs of P2P-TV systems and P2P file sharing applications deeply different.

When building a mesh-based P2P-TV system, two are the key features that must be put in place: *i)* the algorithms adopted to build and maintain the overlay topology [1], [2], [3], [4], [5], [6], *ii)* the algorithms employed to trade chunks [7], [8], [9]. We explicitly focus on the first problem, decoupling its performance from the choice of chunk trading algorithm. Most of the previous works have mainly a theoretical flavor, where performance analysis has been carried out in rather idealized scenarios, by means of simulations or analytical models [3], [4], [5], [6]. Few works undergo implementation and present actual experiments, and even those are usually limited to few tens of peers [10], [11]. See also Sect. VIII for a more complete presentation of the related works.

The aim of this paper is to fill this gap: by using the PeerStreamer P2P-TV application, we present a comprehensive and purely experimental benchmarking of different strategies for the construction and the maintenance of the overlay topology for P2P-TV systems. We thoroughly evaluate different policies, assessing the impact of signaling, measurements, implementation issues, etc. Both synthetic scenarios and PlanetLab experiments are proposed. The first allow us to gauge properties of the different algorithms in increasingly adversarial scenarios. In such scenarios we have the full control of all network parameters, thus they form a scientific and reproducible benchmarking set. The second permit us to further check the validity the previous conclusions into the wild Internet, where natural uncertainty has to be faced.

The system design we present is fully distributed. No centralized tracker or 'oracle' is assumed, and peers collaborate to find the best configuration to use. Topology construction algorithms are based on *selection* and *replacement* criteria according to which each peer chooses the peers it would like to download chunks from. This turns to be very simple and easy to implement. A blacklist-like hysteresis prevents peers to select neighbours previously discarded due to poor performance. Overall, we explore 12 different combinations of criteria (24 if blacklisting is enabled), based on metrics such as Round Trip Time (RTT), upload capacity, number of received chunks, etc. Their performance is thoroughly benchmarked by running fairly large scale experiments (thousands of peers) under different system and network conditions. Measurements are performed considering first traditional Quality of Service

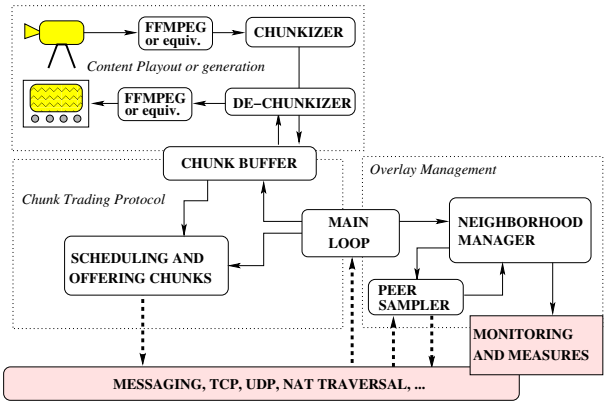

Figure 1. PeerStreamer peer architecture.

- QoS - metrics (e.g., frame loss, delivery delay, etc.) to prune those combination that perform poorly. Afterwards, we consider the Quality of Experience (measured by the SSIM - Structural Similarity Index - [12]) to include all elements in the P2P-TV video distribution chain. Our results have been collected during large experimental campaigns which totally amount to more than 1500 hours of tests.

While the intuition and ideas of our algorithms are simple and well understood in the community, their actual implementation and experimental validation constitute a major step toward the engineering of large scale P2P-TV live streaming systems. As such, the guidelines and results presented in this paper may be useful for researchers, designers and developers interested in P2P-TV application. Our tests show that even simple improvements to a random-based policy for the overlay construction may lead to significant QoE enhancement. Finally, we highlight that the software used in this paper is released as Open Source and includes all the components necessary to build a fully functional P2P-TV system including video transcoding at the source and play-out at clients.

A preliminary version of this paper was presented in [13]. In this version we include an additional scenario devised to bring the system to its limits, and we extend the experiments using PlanetLab. Finally, we introduce some theoretical considerations to support algorithm design choices.

The rest of the paper is organized as follows. We start describing the design and the implementation of our P2P-TV application and its overlay management module in Sect. II. We present the family of algorithms for the overlay construction in Sect. III. In Sect. IV we describe the configuration of the testbeds employed for our experiments. Sect. V presents results of experiments conducted in both a controlled environment and PlanetLab. Finally, we discuss the related work in Sect. VIII, and we conclude in Sect. IX.

II. PeerStreamer Description

Empowering this work is PeerStreamer[1], an Open Source P2P-TV client that stems from the developments and research of the NAPA-WINE project [14], whose overall architecture and vision are described in [15]. PeerStreamer leverages GRAPES [16], a set of C libraries implementing blocks that enables building P2P-TV applications with almost arbitrary characteristics, thus allowing for experimental comparison of different choices to be done efficiently. Fig. 1 describes the logic and modular organization of PeerStreamer. The overlay management, the focus of this paper, is detailed in Sect. II-B, while in the following we sketch the high level organization of the other application components.

A. PeerStreamer Architecture

PeerStreamer is based on a chunk-based diffusion. Each peer offers a selection of the chunks they own to some peers in their neighborhood. The receiving peers acknowledge the chunks it is interested in, thus avoiding multiple transmissions of the same chunk to the same peer. The negotiation and chunk transmission phase is based on signaling exchanges with "Offer" and "Select" messages. For chunk scheduling, Offers are sent to neighbors in round-robin. They contain the buffer-map of the most recent chunks the sender possesses at that time. After receiving an Offer, a peer selects one chunk based on a "latest useful" policy, and sends back a Select message. This has been proven optimal for streaming systems with centralized and distributed scheduling associated to specific peer choices in [7], [9]. The *number of offers* per second a peer sends plays a key role in performance. Intuitively, it should be large enough to fully exploit the peer upload capacity, but it must not be too large to cause the accumulation of chunks to be transmitted adding queuing delay prior to chunk transmissions. We adopt Hose Rate Control (HRC) proposed in [17] to automatically adapt the number of offers to both peer upload capacity and system demand. Simpler trading schemes are less performing and can hide the impact of the overlay on the overall system performance.

The source is a standard peer, but it does not participate in the Offer/Select protocol. It simply injects copies (5 in our experiments) of the newly generated chunk into the overlay. It implements a *chunkiser* to process the media stream (e.g., a live stream coming from a DVB-T card, or from a web-cam). The chunking strategy used in PeerStreamer is chosen to avoid mingling its effects with the topology-related ones: one-frame is encapsulated into one-chunk to avoid that a missing chunk would impair several frames due to, e.g., missing frame headers. The chunkiser is implemented using the ffmpeg libraries[2], so that several different codecs (e.g., MPEG, theora, H.264, etc.) are supported. Receiving peers, instead, implement a de-chunkiser, which reads from the local chunk buffer and pushes the chunks in the correct sequence to the play-out system.

The main loop (at the center of Fig. 1) implements the global application logic. It is responsible for the correct timing and execution of both semi-periodic tasks, e.g., sending new offers, and asynchronous activities, e.g., the arrival of a chunk or signaling message from the messaging layer.

PeerStreamer architecture is completed by the "messaging" and "monitoring and measures" modules. The messaging module is a network abstraction layer that frees the application from all details of the networking environment, e.g., the

[1] Available at http://www.peerstreamer.org

[2] http://www.ffmpeg.org

presence of NAT, middle-boxes and other communication details. It offers a connection-oriented service on top of UDP, with a lightweight retransmission mechanism that allows the recovery of lost packets with low retransmission delay.

The monitoring and measures module extracts network information by running passive and/or active measurements [15]. In this paper we rely on the measurements of *i)* end-to-end path delay between peers (e.g., RTT), *ii)* packet loss rate, and *iii)* transmission rate of a peer.

B. Overlay Management

The approach for building the overlay topology in PeerStreamer is fully distributed: each peer builds its own neighborhood following only local measures, rules and peer sampling. The overlay topology is represented by a directed graph in which the peer at the edge head receives chunks from the peer at the edge tail, which is the one sending offers. Each peer p handles thus an "in-neighborhood" $\mathcal{N}_I(p)$ and an "out-neighborhood" $\mathcal{N}_O(p)$. $\mathcal{N}_I(p)$ collects all peers that can send chunks to p (p in-neighbors); $\mathcal{N}_O(p)$ collects all peers that can receive chunks from p (p out-neighbors). Alternatively, $\mathcal{N}_I(p)$ is the set of peers that *offer* p new chunks; while p *offers* its chunks to peers in $\mathcal{N}_O(p)$. Distinguishing between $\mathcal{N}_I(p)$ and $\mathcal{N}_O(p)$ guarantees a larger flexibility in topology management than algorithms imposing the reciprocity between peers. The overlay topology $\mathcal{T}_S$ is then obtained as union of all the edges connecting peers in $\mathcal{N}_I(p)$ to p, i.e.:

$$\mathcal{T}_S = \bigcup_{p \in \mathcal{S}} \mathcal{N}_I(p) \times \{p\} \tag{1}$$

where $\mathcal{S}$ is the set of all the peers in the swarm and the symbol $\times$ denotes the Cartesian product operator[3].

Referring again to Fig. 1, the topology management is split into two separate functions. The *peer sampler* has the goal of providing p with a stochastically good sample of all the peers in $\mathcal{S}$ and their properties; PeerStreamer implements a variation of Newscast [18] for this function. The *neighborhood manager* realizes the task of filtering the most appropriate peers for interaction. Filtering is based on appropriate metrics and measures, and it is the main focus of this paper.

III. Neighborhood and Topology Construction

In PeerStreamer every peer p selects other peers as in-neighbors and establishes a management connection with them. Thus each peer p actively selects in-neighbors to possibly download chunks when building the set $\mathcal{N}_I(p)$. Similarly, p passively accepts contacts from other peers that will form the set $\mathcal{N}_O(p)$ of out-neighbors. There is no limitation to $N_O(p)$[4].

Every peer p manages a blacklist of peers in which it can put peers that were perceived as very poorly performing in-neighbors. Peers in the blacklist cannot be selected for inclusion in $\mathcal{N}_I(p)$. Blacklisted peers are cleared after the expiration of a time-out (set to 50 s in the experiments).

[3]Notice that since $\mathcal{N}_O(p)$ are built passively, they do not contribute to construction of the swarm topology.

[4]In the actual implementation $N_O(p)$ is limited to 200 peers, but the limit is never reached.

The size N_I of $\mathcal{N}_I(p)$ is equal for every peer p: its goal is to guarantee that p has enough in-neighbors to sustain the stream download with high probability in face of churn, randomness, network fluctuations, etc. The size $N_O(p)$ of $\mathcal{N}_O(p)$ is instead a consequence of the filtering functions of the peers that select p as in-neighbor. The goal is to let the dynamic filtering functions of peers $q \in \{\mathcal{S} \setminus p\}$ select $\mathcal{N}_O(p)$ in such a way that the swarm performance is maximized. For example, peers with higher upload capacity should have larger number of out-neighbors than peers with little or no upload capacity [4].

The update of neighborhoods is periodic, maintaining the topology dynamic and variable, so that churn impairment is limited, and the swarm can adapt to evolving networking conditions. In particular, every T_{up} seconds each peer p independently updates $\mathcal{N}_I(p)$ by *dropping* part of the old in-neighbors while *adding* fresh in-neighbors. Two parameters are associated to this scheme: the update period T_{up} and the fraction F_{up} of peers in $\mathcal{N}_I(p)$ that is replaced at every update. The add operation guarantees $\mathcal{N}_I(p)$ has size N_I (if at least N_I peers are known). Overall, the in-neighbor update rate can be defined as

$$R_{\text{up}} = \frac{F_{\text{up}} N_I}{T_{\text{up}}} \tag{2}$$

If not otherwise stated $N_I = 30$, $T_{\text{up}} = 10\,\text{s}$ and $F_{\text{up}} = 0.3$. The latter two values result in a good compromise between adaptiveness and overhead. Their choice is robust, and sensitivity analysis is presented in Sect VI-C.

A. Metrics Driving The Neighborhood Selection

At every update, $\mathcal{N}_I(p)$ is the result of two separate filtering functions: one that selects the peers *to drop*, and another one selecting in-neighbors *to add*. For these filtering functions we consider both simple network attributes such as peer upload bandwidth, path RTT or path packet loss rate, and some application layer metrics, such as the peer offer rate[5] or number of received chunks from an in-neighbor.

Some metrics are static *peer metrics*: once estimated, they can be broadcasted with gossiping messages and are known *a-priori*. Other metrics instead are *path attributes* between two peers and must be measured and can only be used as *a-posteriori* indicators of the quality of the considered in-neighbor as perceived by p.

Both add and drop filtering functions are probabilistic to avoid deadlocks and guarantee a sufficient degree of randomness. Considering a metric, we assign a selection probability w_q to every candidate q as

$$w_q = \frac{m_q}{\sum_{s \in \mathcal{N}_S(p)} m_s} \tag{3}$$

where m_q is the metric of q and $\mathcal{N}_S$ is either $\mathcal{N}_I$ for drop, or the set of candidate in-neighbors for add.

[5]HRC adapt the peer offer rate to peer upload capacity. It can thus be seen as an indirect measure of its available upload bandwidth.

B. Add Filters

We consider the following four criteria to add new in-neighbors:
RND: Neighbors are chosen uniformly at random: $\forall q, m_q = 1$;
BW: Neighbors are weighted according to their upload bandwidth C_q: $\forall q, m_q = C_q$;
RTT: Neighbors are weighted according to the inverse of the RTT between p and q: $\forall q, m_q = 1/RTT_q(p)$; if $RTT_q(p)$ is still unknown, $RTT_q(p) = 1\,\text{s}$[6];
OFF: Neighbors are weighted according to the rate they send offer messages R_q: $\forall q, m_q = R_q$; R_q are advertized by peers.

C. Drop Filters

For what concerns the criteria to select neighbors to be dropped, we consider:
RND: Neighbors are dropped randomly: $\forall q, m_q = 1$;
RTT: Neighbors are dropped with a probability directly proportional to the RTT between p and q: $\forall q, m_q = RTT_q(p)$;
RXC: Neighbors are dropped with a probability proportional to the inverse of the rate at which it transferred chunks to p: $\forall q, m_q = 1/RXC_q(p)$; this metric assigns a quality index related to the in-neighbor ability to successfully transfer chunks to p; $RXC_q(p)$ are evaluated on a window of 3 s.

D. Blacklisting Policies

Finally a peer in $\mathcal{N}_I(p)$ is blacklisted if one of the following criterion is met:
CMR: the ratio of corrupted/late chunks among the last 100 chunks received by p from q exceeds a threshold of 5%;
PLOSS: the packet loss rate from q to p exceed a threshold of 3%; measured over the last 300 packets received;
RTT: $RTT_q(p)$ is greater than 1 s.

Combining add and drop criteria we define 12 different overlay construction and maintenance filters. In the following, we name them stating the "ADD"-"DROP" policies, e.g., BW-RTT for add BW and drop RTT. Sect. V reports results for different resulting combinations. Blacklisting can be superposed (or not) to all of them, and its impact will be studied selectively. We tested also other metrics and combinations, whose results are less interesting. RND-RND is used as a baseline benchmark, as it is a policy based on pure random sampling of the swarm.

E. A few theoretical considerations

Depending on the distributed neighbor selection strategy adopted by peers, the resulting overlay topology $\mathcal{T}_S$, as defined in (1), may exhibit fairly different macroscopic properties, especially when $|\mathcal{S}|$ grows large. Some considerations based on elementary notions on random graphs will help to better understand the effect of different neighbors' selection policies.

On the one hand, it would be highly desirable to obtain an overlay topology $\mathcal{T}_S$ with very good structural properties, such as high node resilience, short diameter and large conductance. Observe, indeed, that the graph theoretical concept of node resilience can be almost immediately re-interpreted as resilience to peer churning, while conductance/diameter properties of the graph have been recently shown to be tightly related with the minimum time needed to spread information over a graph with homogeneous edge capacities [19].

On the other hand, just structural graph properties of $\mathcal{T}_S$ may not be sufficient to represent well how good the topology is, since peers upload bandwidth may be highly heterogeneous and edge latencies (i.e., RTTs) may play an important role on performance. Neighbor selection policies should favour "local" edges (i.e., edges between peers that are geographically and/or network-wise close) reducing edge latencies (with the positive side effect of localizing the traffic). Furthermore they should not build topologies with a constant degree of connectivity, but privilege strategies that lead to high connectivity degrees for high bandwidth peers to exploit their upload resources at the best. As it will be evident in the following, to achieve high and reliable performance it is important to properly balance the previous ingredients, aiming at obtaining a topology $\mathcal{T}_S$ that blends good global (topological) properties, with a bias toward the exploitation of local properties like vicinity and bandwidth availability.

The simplest possible selection strategy is **RND-RND**. It is known to produce an overlay topology $\mathcal{T}_S$ with good global properties, and for these reasons it is adopted by several P2P systems [20]. Indeed the overlay topology resulting by the adoption of **RND-RND** can be modeled as an n-regular (directed) random graph instance with $n = N_I$. Such graphs have high resilience (whenever n is large than few units), logarithmic diameter, and high conductance. The limit of **RND-RND** consists in the fact that it completely ignores peer attributes (such as location and upload bandwidth).

Policies that select/drop neighbors according to the peer-to-peer distances (i.e., RTTs) better localize the neighborhoods. Of course the application can profit from neighborhoods localization since message latencies/chunk transfer times are potentially reduced. However, a too strict localization of neighborhoods can jeopardize the global properties of $\mathcal{T}_S$, with detriment of the application performance as already observed in [5], [21], [22]. The danger of an excessive localization of neighborhoods can be easily explained from a graph theoretical perspective; indeed geographical graphs (i.e., graphs in which edges are established only between nodes which are close with respect to some parameter) exhibit poor diameter/conductance properties (the graph diameter, for example, scales as the square root of number of nodes in bi-dimensional geographical graphs). Furthermore, when nodes are non homogeneously distributed over the domain, the topology may become disconnected (conductance equal to zero) even if the average connectivity is high. Thus, localization must be pushed only up to a given point, guaranteeing the presence of a non-marginal fraction of chords (i.e., non local edges). The presence of a significant fraction of random chords guarantees the resulting overlay $\mathcal{T}_S$ to exhibit "small word" properties, i.e., good diameter/conductance properties. At the same time the massive presence of local edges potentially permits to reduce the negative effects of large latencies. Observe that

[6] $RTT_q(p)$ are locally cached at p so that they may be available a priori. Active measurements could also be used to quickly estimate the RTT.

Table I
NUMBER OF PCS PER SUBNET.

Subnet	1	2	3	4
Number of PCs	43	63	60	38

Table II
RTTS IN ms BETWEEN SUBNETS OF PEERS.

	1	2	3	4
1	20 ± 10%	80 ± 10%	120 ± 10%	160 ± 10%
2	80 ± 10%	20 ± 10%	140 ± 10%	240 ± 10%
3	120 ± 10%	170 ± 10%	20 ± 10%	200 ± 10%
4	160 ± 10%	240 ± 10%	200 ± 10%	20 ± 10%

to guarantee a significant amount of chords our **RTT** based selection/dropping filter are probabilistic.

The adoption of policies that select peers taking into account direct or indirect measurements of the peers' upload bandwidth (such as **BW** or **OFF**) make the average out-degree proportional to the peer upload bandwidth. We observe that these policies directly influence the conductance of the resulting $\mathcal{T}_S$, as they increase average weight of edges. This is due to a global policy that aims at an n-regular directed graph, so that the number of edges in the topology is roughly constant. Having more edges insisting on high bandwidth links and less on low bandwidth ones increases the average weight.

It is harder, instead, to predict the effect on $\mathcal{T}_S$ of the adoption of **RXC** dropping filters. In this case, we attempt to correlate neighbor filtering decisions to an estimate of the real neighbors' "quality", with the goal of preserving the best in-neighbors (peers that have shown to be helpful in retrieving chunks in the recent past) while discarding bad in-neighbors (peers that were scarcely helpful). It can be argued, however, that also in this case the outcome of the strategy is an increased topology conductance, as the actual or a-posteriori weight of edges is proportional to the amount of chunks correctly transferred along it. Again we ensure that a significant amount of random edges are in $\mathcal{T}_S$ by adopting probabilistic filtering, so as to guarantee good global properties.

IV. TEST-BED CONFIGURATION

We need to benchmark the different algorithms in a known and reproducible scenario. To this aim, we run experiments in a possibly complex, but fully controlled network to avoid fluctuations and randomness due to external impairments. The test-bed is built in labs available at Politecnico di Torino, with 204 PCs divided in four different subnets. Table I shows the number of PCs in each subnet. We used `tc`, the standard Linux Traffic Controller tool, together with the `netem` option to enforce delay and packet dropping probability when needed. The chosen RTT distribution is described in Table II. The upload bandwidth is limited by the application itself, exploiting the feature of a simple leaky bucket (its memory being 10MB) to limit the application data rate to a given desired value. Peer upload capacities C_p are shown in Table III. Configurations in Tables II and III have been designed to resemble a world-wide geographic scenario, where peers are distributed over continents (clusters), and they rely on different kinds of access technologies, i.e., ADSL or FTTH interfaces, that provide different up-link capacity. Those configurations

Table III
CHARACTERISTICS OF PEER CLASSES.

Class	Upload Bandwidth	Percentage of Peers
1	5 Mb/s ± 10%	10 %
2	1.6 Mb/s ± 10%	35 %
3	0.64 Mb/s ± 10%	35 %
4	0.2 Mb/s, ± 10%	20 %

are not meant to be representative of any actual case, but rather they are instrumental to create benchmarking scenarios with different properties. Each PC runs 5 independent instances of PeerStreamer simultaneously, thus, a swarm of 1020 peers is built in every experiment, if not otherwise stated. The source peer runs at an independent server (not belonging to any of the subnets). It injects in the swarm 5 copies of each newly generated chunk, corresponding to roughly 6 Mbit/s.

The well known *Pink of the Aerosmith* video sequence has been used as benchmark. The nominal sequence length corresponds to 200s, with a time resolution equal to 25 frame/s. The sequence is looped for a total stream duration of about 20 min. After the initial 12 min of experiment, each peer starts saving on local disk a 3 min long video that we use to compute QoE metrics.

We selected the H.264/AVC codec to encode the video sequence. A hierarchical type-B frames prediction scheme has been used, obtaining 4 different kinds of frames that, in order of importance, are: IDR, P, B and b. The GOP structure is IDR$\times$8 {P,B,b,b}. The nominal video rate of the encoder r_s is 1.2 Mb/s if not otherwise specified. This corresponds to a system load $\rho = 0.9$ – defined as $\rho = r_s/E[C_p]$ where $E[C_p] = 1.32$ Mbit/s is the average upload bandwidth of peers.

The source node generates a new chunk at regular time, i.e., every new frame. The chunk size is instead highly variable due to the encoded video characteristics. Each peer implements a chunk buffer of 150 chunks. Given the one-frame⇔one-chunk mapping, and 25 fps of the video, this corresponds to a buffer of 6s, i.e., the play-out deadline is only 6 s.

A. Network Scenarios

The generic setup described above is used as a base for three different scenarios to evaluate significant situations. The first scenario, ***G_Homo*** hereafter, is geographically homogeneous: the distribution of the peers of different C_p classes is the same in any area, so that there is the same distribution of bandwidth everywhere. This scenario is useful to understand the fundamental behavior of different neighborhood filtering strategies.

The second scenario, ***G_Bias*** hereafter, assumes that bandwidth rich peers (Class 1) are all concentrated in a single subnet. This situation is particularly challenging for a topology management system that tries to localize traffic to reduce the network footprint of the application.

The third scenario, ***G_Lossy*** hereafter, is again geographically homogeneous, but the long-haul connections between the subnets 1–3, 1–4, 2–3, 2–4 are subject to packet loss with probability $p = 0.05$, while the intra-subnet links and the links between 1–2 and 3–4 are lossless. This situation is useful to understand if black-listing can really help in building

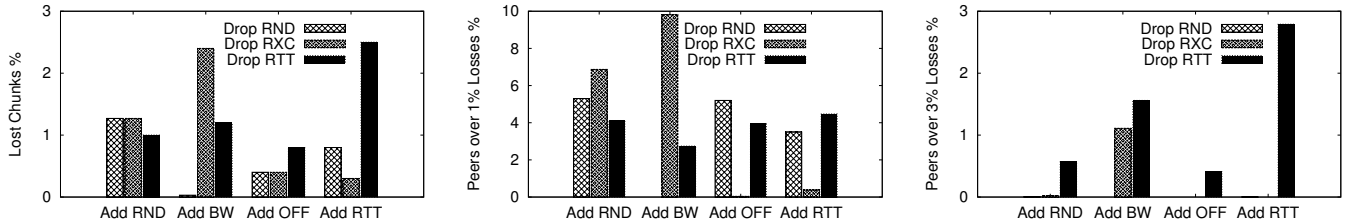

Figure 2. Frame loss for different strategies in *G_Homo* scenario: F_{loss} (average) (left), percentage of peers whose $F_{loss}(p) > 0.01$ (center), percentage of peers whose $F_{loss}(p) > 0.03$ (right).

better topologies, or if its use should be limited to isolate misbehaving and malicious nodes.

The fourth and final scenario, ***G_Adver*** hereafter, combines together *G_Bias* and *G_Lossy* in such a way that large bandwith peers are amassed in the subnet 1, while the four subnets experience the same packet loss configuration on long-haul links described above. This case, even if adverse, represents an interesting challenge for those strategies that seek the tradeoff between location and bandwidth awareness, and it is useful to detect possible conflicts with the black-listing functionality.

Finally, churning of peers is modeled: a fraction P_{no-ch} of peers never leaves the system, while $P_{ch} = 1 - P_{no-ch}$ churning peers have a permanence time uniformly distributed between 4 and 12 min. To keep the number of peers constant, once a churning peer has left the system, it will be off for an average time equal to 30 sec before re-joining the swarm (with a different ID, i.e., as a new peer).

B. Performance Indices

As performance indices to assess the QoE, for each peer p, we consider the *frame loss probability*, $F_{loss}(p)$, and the SSIM (Structural Similarity Index), $S_{ssim}(p)$, a well-known method for measuring the similarity between two images in the multimedia field [12]. Given the highly structured organization of the video streams, the degradation of the received video quality becomes typically noticeable for values of $F_{loss}(p)$ higher than 1%, while loss probability of a few percent (3-4%) significantly impair the QoE. In the following, we report both average frame loss, $F_{loss} = E_p[F_{loss}(p)]$, and the percentage of peers that suffer $F_{loss}(p)$ larger than 1% and 3%, respectively.

Performance however should also take into account the cost for the network to support the application. As *network cost* ζ we consider the average of the distance traveled by information units. Formally, let $b_q(p)$ the number of bits peer p received from peer q; the peer p network cost $\zeta(p)$ is computed as

$$\zeta(p) = \frac{\sum_q RTT_q(p) b_q(p)}{\sum_q b_q(p)} \tag{4}$$

while the average network cost is $\zeta = E_p[\zeta(p)]$.

V. Controlled Environment Experiments

A. G_Homo Scenario

We start considering the case in which the distribution of C_p is geographically homogeneous.

The left-hand plot in Fig. 2 shows the average frame loss probability experienced by different policies, while center and right-hand plots report the percentages of peers that experienced $F_{loss}(p) > 0.01$ and $F_{loss}(p) > 0.03$, respectively.

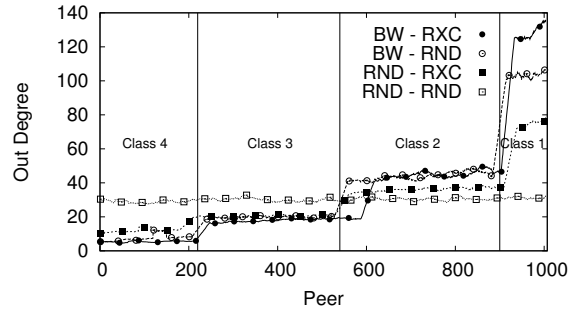

Figure 3. Out-degree distribution of peers, *G_Homo* scenario.

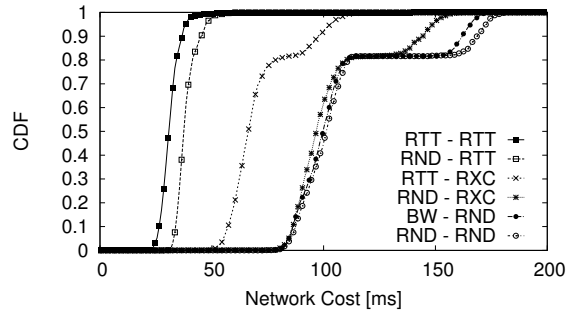

Figure 4. CDF of the distance traveled by information units, *G_Homo* scenario.

RND-RND is the reference, and we immediately observe that the other algorithms modify the loss *distribution*, i.e., they can have a different impact on different percentiles. For instance BW-RTT improves the average loss rate and the percentage of peers with $F_{loss}(p) > 0.01$, but at the expense of the percentage of peers with bad quality ($F_{loss}(p) > 0.03$), while RTT-RTT improves the number of peers with $F_{loss}(p) > 0.01$, but both the average and the percentage of peers with bad quality ($F_{loss}(p) > 0.03$) are worse.

In general, the use of policies sensitive to peer bandwidth (BW and OFF for adding and RXC for dropping) appear to be the more effective in reducing the losses. However the behavior of BW-RXC for which F_{loss} tops at 2.5% indicates that using a single metric for selecting the neighborhood can be dangerous. BW-RXC biases too much the choices toward high bandwidth peers, which become congested and are not able to sustain the system demand. To better grasp these effects, Fig. 3 reports the smoothed[7] histogram of the out-degree $N_O(p)$.

[7]The distribution of $N_O(p)$ inside classes is binomial as expected from theory. This distribution results in a large noisiness of the plot, so we apply a smoothing window of length 30 in plotting, basically showing the average N_O in each class.

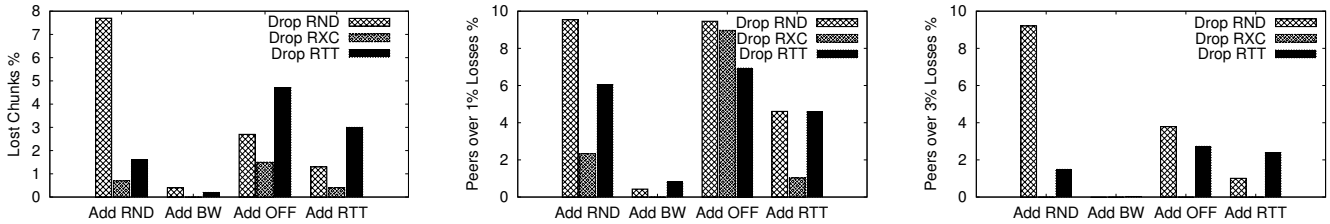

Figure 5. Frame loss for different strategies in *G_Homo* scenario with $N_I = 20$: F_{loss} (average) (left), percentage of peers whose $F_{loss}(p) > 0.01$ (center), percentage of peers whose $F_{loss}(p) > 0.03$ (right).

Observe that $N_O(p)$ of peers belonging to different classes is significantly different as long as bandwidth aware policies are adopted; out-degrees are instead independent for RND-RND as expected. In principle it would be desirable to have an out-degree of a peer proportional to its up-link bandwidth. This is roughly achieved by adopting BW-RND policy. Under BW-RXC, instead, the degree distribution depends too much on C_p. As a result, high bandwidth peers tends to be oversubscribed while medium and low bandwidth peers may be underutilized.

Policies sensitive to RTT perform well in the considered scenario, with the exception of RTT-RTT, which is too aggressive in strictly selecting the closest in-neighbors. Indeed, as observed in [5], policies that force a too strict localization of traffic induce performance degradations due to poor topological properties of the swarm. To complement previous information Fig. 4 reports the Cumulative Distribution Function (CDF) of network cost $\zeta(p)$. As expected, RTT aware policies significantly reduce this index thanks to their ability to select in-neighbors within the same area.

Remark A - As a first consideration, we can say that: *i)* bandwidth aware policies improve the application performance; *ii)* RTT aware policies reduce the network cost without endangering significantly the video quality if applied to add peers; when used to drop peers, however, RTT poses significant bias impairing QoE; *iii)* the preference toward high bandwidth peers/nearby peers must be tempered to achieve good performance. The policy RTT-RXC improves quality and reduces the network cost at the same time, offering the best trade-off in this scenario. Interestingly, this policy is also easy to be implemented, since it requires to measure simple and straightforward metrics. Bandwidth aware schemes offers better QoE performance, at the cost of more cumbersome available capacity estimation.

B. G_Homo with Smaller N_I

We consider the same network scenario but we set $N_I = 20$. This is a more critical situation where choosing the good in-neighbors is more important. The value of N_I is related with the signaling overhead which increases with N_I, so having small neighborhood is desirable. However, a too small N_I would impair the availability of chunks.

Results are plotted in Fig. 5 (the y-scales in Figs. 2 and 5 are different for readability reasons, and this is the reason why at first sight some policies seem to perform better with a smaller N_I). The performance of RND-RND significantly degrades in this case. The reason is that the out degree of Class 1 peers under RND-RND is often not enough to fully exploit their bandwidth. Bandwidth aware strategies, instead, successfully adapt $N_O(p)$ to C_p maintaining high performance. Also RTT-RND and RTT-RTT, which are bandwidth unaware, perform better than RND-RND, since RTT-aware selection policies reduce the latency between an offer and the actual chunk transmission that follows it, helping in exploiting the peer's bandwidth. Results for network cost are similar to those in Fig. 4 and are not reported for the sake of brevity.

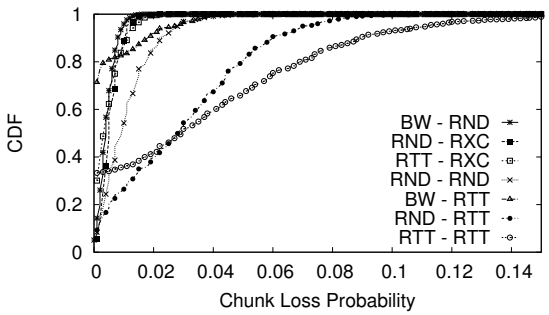

Figure 6. CDF of the frame loss probability for four different strategies, *G_Bias* scenario.

Remark B - Random selection policies, which are widely employed by the community as baseline and in the wild [20], are robust, but perform poorly if the number of peers in the neighborhood is small: all peers suffer 8% of frame loss, i.e., practically making it impossible to decode the video. As already seen with $N_I = 30$, the policy that combines bandwidth and RTT awarenesses (RTT-RXC) definitely improves both performance and network costs. Similarly, wisely selecting high-capacity in-neighbors is vital, as testified by the excellent performance of add BW policies.

C. G_Bias Scenario

Maintaining unchanged the C_p distribution, we localize all high bandwidth peers in geographical area 1. This scenario, in principle, constitutes a challenge for the policies that try to localize traffic. Indeed as side effect of the localization we can potentially have a "riches with riches", "poors with poors" clusterization effect that may endanger the video quality perceived by peers in geographical regions other than 1.

Fig. 6 reports the CDF of $F_{loss}(p)$ for the strategies performing better in the *G_Homo* scenario, plus the benchmark RND-RND. In this case if RTT is the only metric used as in RTT-RTT, the performance degrades unacceptably, and peers in area 1 are in practice the only one receiving a good service. In general, any policies based on drop RTT perform poorly. Strategies RTT-RXC, RND-RXC and BW-RND perform similarly; however, the only policy that can also

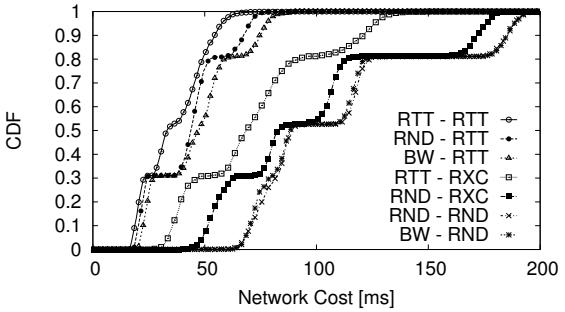

Figure 7. CDF of distance traveled by information units, *G_Bias* scenario.

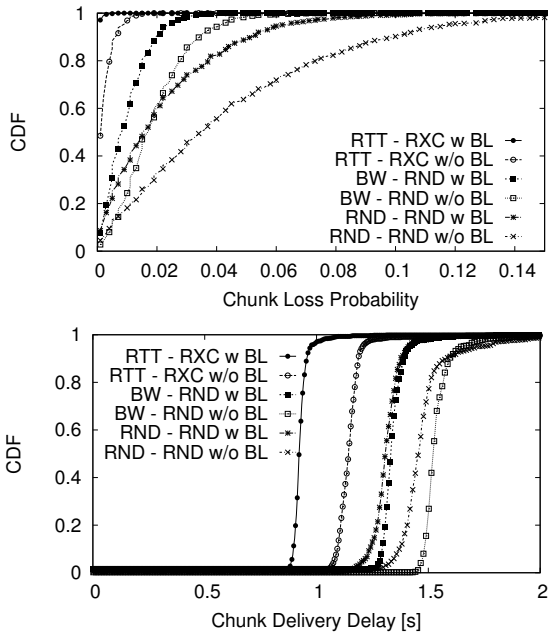

Figure 8. CDF of chunk loss probability (top) and CDF of chunk delivery delays (bottom) for six different strategies with and without adopting blacklist mechanism in *G_Lossy* scenario.

reduce the network cost is RTT-RXC, as shown in Fig. 7 that reports the CDF of $\zeta(p)$.

Remark C - This result essentially proves that also in *G_Bias* scenario it is possible to partially localize the traffic without endangering the video quality perceived by the user, as long as RTT awareness is tempered with some light bandwidth awareness, as in RTT-RXC. Interestingly, the RTT driven policies perform much better if the RTT is used to *add* peers rather than to *drop* peers. Indeed, in this latter case, aggressively dropping far away, but high capacity, in-neighbors penalizes peers which are located in areas where little high capacity peers can be found.

D. G_Lossy Scenario

We consider another scenario in which large bandwidth peers are uniformly distributed over the four subnets, but packet losses are present in some long haul connections.

Fig 8 plots the CDF of frame losses (top) and the CDF of chunks delivery delays (bottom) for the selected policies. Blacklisting improves the performance of every policy. RTT-RXC emerges again as the most performing policy and with blacklisting practically all peers are able to receive all chunks.

Table IV
AVERAGE FRACTIONS OF INCOMING TRAFFIC FOR CLUSTER 2.

	1 - good	2 - local	3 - bad	4 - bad + far
RND - RND w/o BL	0.23	0.32	0.28	0.15
RND - RND w BL	0.28	0.34	0.24	0.12
BW - RND w/o BL	0.22	0.35	0.27	0.14
BW - RND w BL	0.23	0.36	0.24	0.13
RTT - RXC w/o BL	0.12	0.68	0.11	0.07
RTT - RXC w BL	0.13	0.70	0.09	0.05

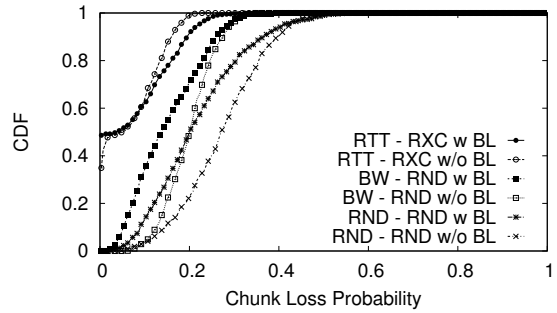

Figure 9. CDF of chunk loss probability for six different strategies with and without adopting blacklist mechanism in *G_Adver* scenario.

This is an excellent result, since the system is facing a very challenging scenario while working with a load of 0.9.

Benefits of the blacklisting mechanism are confirmed by Table IV that reports the normalized volume of incoming traffic for peers in cluster 2 from peers in all clusters. Keeping in mind that in *G_Lossy* scenario peers belonging to cluster 2 experience lossy paths from/towards peers in cluster 3 and 4 (as explained in Sec. IV), it is easy to see that volumes of incoming traffic from cluster 3 and 4 are nicely reduced thanks to blacklisting mechanism.

Remark D - Blacklisting can play a significant role to avoid selecting lossy paths. Indeed, exploiting the blacklist mechanism every peer should identify and abandon poorly performing peers, biasing the neighborhood toward good performing in-neighbors. This effect reinforces policies that naturally bias the selection of neighbor peers employing peer quality. RND-RND, BW-RND and RTT-RXC have emerged as the most promising criteria (RND-RND being the baseline benchmark). RTT-RXC with blacklisting is shown to guarantee excellent performance to all peers even in this almost adversarial scenario.

E. G_Adver Scenario

Fig. 9 plots the CDF of frame losses for different policies in *G_Adver* scenario when the system load $\rho = 0.9$. Policies which showed best performance in *G_Lossy* where considered. Comparing Fig. 9 with Figs. 6 and 8 (top plot), it is immediately evident how combining together *G_Bias* and *G_Lossy* configurations has a negative impact on performance. However, even in this difficult context, RTT-RXC policies still shows the lowest chunk loss probability, confirming to be the best choice. As seen in Sec. V-D, also in this case blacklisting improves the performance, especially of those filtering strategies which do not exploit location-awareness, such as RND-RND and BW-RND. This instead does completely for RTT-RXC, and induce to think that the tradeoff between

location and bandwidth awareness is enough to provide acceptable performance in this challenging scenario. Indeed, given the peculiar configuration of *G_Lossy*, choosing in-neighbors based on their location represents a sufficient criterion to avoid lossy links, so that blacklisting does not introduce any further benefit.

Remark E - Blacklisting confirms to be in general a useful help in the filtering process, letting peers drop lossy connections, especially when no location-based filtering strategy is adopted.

VI. Video Performance Evaluation

A. Video performance versus load

We now summarize the results by aessessing the actual average QoE by reporting S_{ssim} for different policies and different system loads. We consider *G_Lossy* and *G_Adver* scenarios, and we swipe r_s from 0.6 Mb/s to 1.4 Mb/s. Recall that $E[C_p] = 1.324$ Mb/s.

Fig. 10 shows average S_{ssim} considering RND-RND, BW-RND and RTT-RXC with and without blacklisting. SSIM is a measure of the distortion of the received image compared against the original source (before encoding and chunkization). It is a highly non linear metric in decimal values between -1 and 1. Negative values correspond to negative images, so are not normally considered at all. Values above 0.985 are typically considered of excellent quality. SSIM has been computed considering the video between min. 12 and 13 (60x25 frames) received by 200 peers (50 for each class), and then averaging among all of them.

The EVQ (Encoded Video Quality) curve in the plot is the reference value for the encoding rate and it obviously increases steadily as r_s increases. For all policies, when the system load is small $\rho << 1$, average S_{ssim} increases for increasing r_s thanks to the higher quality of the encoded video. However, as ρ approaches 1, different policies behave differently: S_{ssim} rapidly drops due to missing chunks which impair the quality of the received video, but the degradation is highly influenced by the topology. Notice how RTT-RXC scheme outperforms RND-RND and BW-RND for every value of r_s. Fig. 10 also shows the benefits of the blacklist mechanism for every scheme.

Similarly, Fig. 11 shows average S_{ssim} considering RND-RND, BW-RND and RTT-RXC in the *G_Adver* scenario. Also in this case we compare all policies with and without blacklisting functionality. As expected, the adversarial scenario makes it more challenging to acheive a good QoE, even for $\rho << 1$: i) RTT-RXC shows definitely the best performance, even if worsened by the adversarial conditions of this scenario, ii) enabling blacklisting induces a great gain for not location-aware policies, i.e. RND-RND and BW-RND, but in this extreme scenario this does not hold for strategies which aim at minimizing propagation delays, i.e. RTT-RXC.

Remark F - RTT-RXC with blacklisting guarantees optimal QoE for $\rho < 1$ whereas RND-RND policies are not able to guarantee good QoE for $\rho > 0.75$.

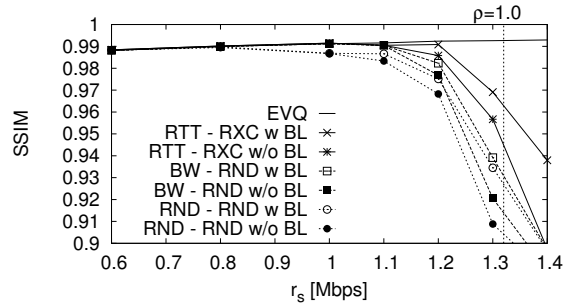

Figure 10. S_{ssim} (average) index when varying video rate. *G_Lossy* scenario.

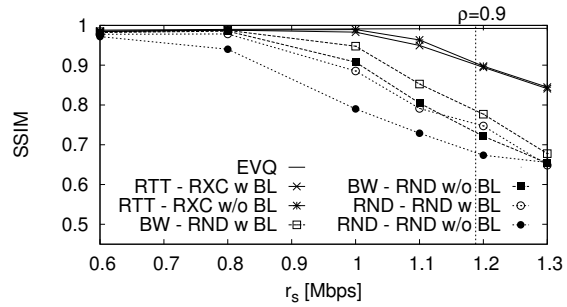

Figure 11. S_{ssim} (average) index when varying video rate. *G_Adver* scenario.

B. Scaling with swarm size

Considering again *G_Homo* scenario, we study how the system scales when increasing the size of the swarm N from 200 to 2000 peers. Due to the lack of space, we only report in Table V the average S_{ssim} for three different values of N. RND-RND and RTT-RXC schemes have been adopted as benchmark. Transmitted video was encoded at $r_s = 1.2$ Mb/s, i.e. system load $\rho = 0.9$. The simple bandwidth-aware scheme, RTT-RXC, always ensures better performance with respect to RND-RND, i.e. the average S_{ssim} improves from 0.8 to 0.99, a remarkable gain. Increasing N has a negligible impact on performance, especially when the smart RTT-RXC policy is adopted. Indeed, in RND-RND case, the topology overlay evolution causes more random results due to the totally random nature of the scheme.

C. Sensitivity to Update Rate with Churning

We investigate what are the best trade-off values for the frequency to update the incoming neighborhood, R_{up} as defined in (2).

Consider a *G_Homo* scenario with P_{churn} fraction of peers that join and leave the swarm. In Fig. 12 we report the S_{ssim} (computed and averaged over peers that never leave the system) when varying the rate of update R_{up}. In particular, we fix $N_I = 30$, $F_{\text{up}} = 0.3$ and change $T_{\text{up}} \in [2, 100]$ s accordingly. For this case we adopted scheme RTT-RXC and

Table V
Average S_{ssim} when increasing the number of involved peers N. *G_Homo* scenario, $r_s = 1.2$ Mb/s.

N	204	612	1040	1428	1836	2080
RND - RND	0.858	0.812	0.829	0.783	0.799	0.799
RTT - RXC	0.984	0.981	0.988	0.979	0.988	0.991

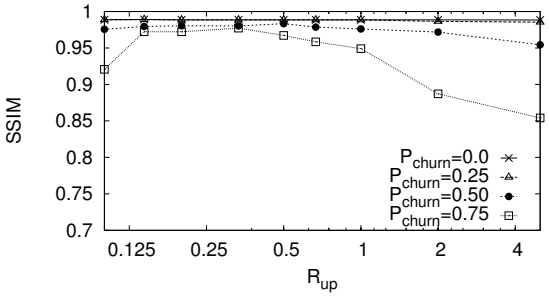

Figure 12. Average S_{ssim} vs R_{up} for different fractions of churning peers P_{churn}. Scheme RTT-RXC in *G_Homo* scenario with 1000 peers and $r_s = 0.8$ Mb/s.

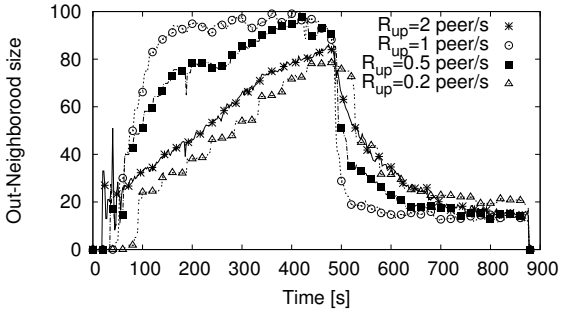

Figure 13. The evolution during time of the average outgoing neighborhood size setting different R_{up} values. Scheme RTT-RXC in *G_Homo* scenario with $r_s = 1.0$ Mb/s.

$r_s = 0.8$ Mb/s. The plot shows that the system is very robust to different R_{up} values. Only under stressed scenarios, such as for $P_{churn} >= 0.5$, R_{up} becomes critical: too high R_{up} does not let the swarm achieve a stable state, impairing performance. On the other hand, too low R_{up} induces peers to react slowly to sudden changes brought by churning peers.

We considered the *G_Homo* scenario again, but forcing all high-bandwidth peers to experience an abrupt up-link bandwidth reduction from 5 Mb/s to 0.64 Mb/s (on average) at time 480 s from the beginning of the video transmission. While this scenario is rather artificial, it allows to gauge the reactiveness of the topology to such abrupt changes. We consider the RTT-RXC scheme. Fig. 13 reports the evolution over time of the average size of the outgoing neighborhood N_O of class 1 peers. Different values of in-neighborhood update rate R_{up} are considered. Two observation holds: first, smaller values of R_{up} slow down system reactiveness. However, too large values, e.g., $R_{\text{up}} = 2$ peer/s, impair the performance as well: in this case, peers have not enough time to collect significant measurements about the in-neighbor "quality" (amount of received chunks), and thus find it difficult to distinguish "good" from "bad" in-neighbors. Also in this case $R_{\text{up}} = 1$ peer/s setup represents a good trade off.

Remark G - Fast topology updates allow the overlay topology *i)* to react quickly to changes in the network scenario and *ii)* to prune quickly peers which left the system in, e.g., heavy churning conditions. However, too fast updates introduce instability in the overlay construction process, driving peers to never achieve a stable incoming neighborhood, and thus leading to bad system performance. The best trade-off R_{up} value is $R_{\text{up}} = 1$ peer/s, i.e., $T_{\text{up}} = 10$ s.

Table VI
CHARACTERISTICS OF PEER CLASSES IN THE PLANETLAB EXPERIMENT.

Class	Upload Bandwidth	Percentage of Peers
1	1.32 Mb/s ± 10%	50 %
2	0.64 Mb/s ± 10%	50 %

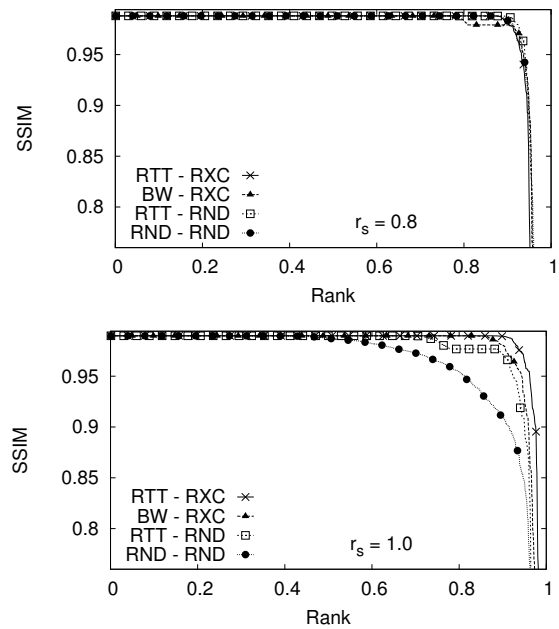

Figure 14. $S_{ssim}(p)$ for $r_s = 0.8$ Mb/s and $r_s = 1.0$ Mb/s for PlanetLab experiments.

VII. PLANETLAB EXPERIMENTS

We now present similar experiments on PlanetLab. We selected all active PlanetLab nodes, excluding those that had configuration issues or severe reachability problems. As a result, we obtained a set of 449 nodes scattered worldwide. No artificial latency or packet loss were imposed, so that connections among these nodes reflect natural Internet conditions.

In order to achieve a scenario similar to the experiments of the previous section, peer upload capacity has been limited by the PeerStreamer embedded rate limiter. Observe that this only guarantees an upper bound to the actual available peer upload bandwidth which may be lower due to competing experiments running on the same PlanetLab node or due to other bottlenecks on the access links of the node. For this reason, contrary to previous experiments, we did not use a 5 Mbit/s class, and restricted the scenario to two classes as shown in Table VI: half of the peers have 2 Mbit/s at their up-

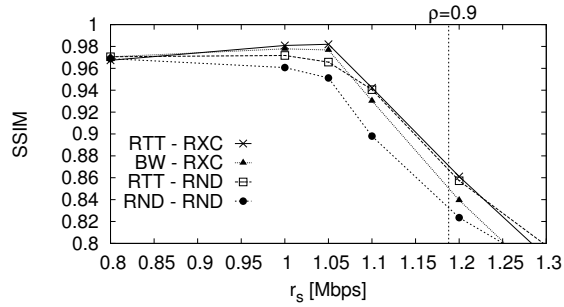

Figure 15. S_{ssim} (average) index when varying video rate for PlanetLab experiments.

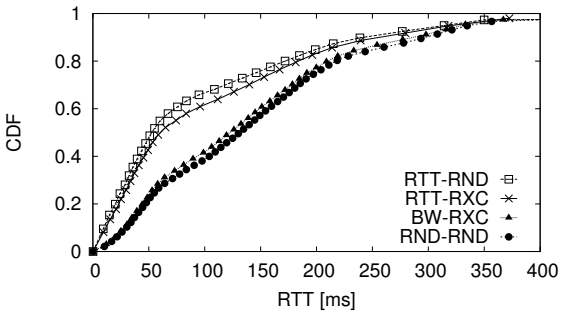

Figure 16. CDF of the distance traveled by information units. PlanetLab experiments.

link, and 0.64 Mbit/s the other half. Average upload capacity results to have an upper bound of 1.32 Mbit/s, but the actual value largely depends on the status of PlanetLab nodes and their network connection. Blacklisting was active by default for this part of experiments.

Fig. 14 reports each peer's individual SSIM performance, $S_{ssim}(p)$, for $r_s = 0.8$ Mbit/s (top) and $r_s = 1.0$ Mbit/s (bottom). $S_{ssim}(p)$ has been sorted in decreasing values to ease visualization. Each curve represents the average of 10 different runs. Observe that when the amount of system resources is large enough with respect to the video-rate, i.e, when $r_s = 0.8$ Mbit/s (top plot), different schemes for topology management perform rather similarly. Observe, however, that there is always a certain fraction of nodes that cannot receive the video due to congestion at local resources.

Increasing system load, i.e. $r_s = 1.0$ Mbit/s (bottom plot), highlights differences among schemes and confirms results obtained in the controlled environment: random-based policies (RND-RND) perform badly in general; same holds for schemes based on pure proximity that can lead to disconnected topologies and, then, to bad QoE performance (RTT-RND). In fact, the right side of the RTT-RND curve shows that a group of peers (with rank around 0.8) received the video with lower quality than other peers.

A pure bandwidth based policy (BW-RXC) provides good performance, delivering the video in good quality to more peers than the previous two policies. However, the best quality (i.e., the highest portion of peers receiving good quality) is achieved when combining bandwidth-awareness with proximity-based schemes (RTT-RXC).

Thus, this last policy achieves the goal of localizing traffic without impairing performance.

Results depicted in Fig. 14 are confirmed in Fig. 15 which shows average S_{ssim} as a function of video-rate r_s, and thus system load ρ, for the same four policies (RND-RND, RTT-RND, BW-RXC, RTT-RXC) in PlanetLab context. Again RTT-RXC proves to be the most reliable choice, consistently outperforming other policies for each value of r_s.

As we did in Sec. V, also in this case, we complement information about users' perceived performance with statistics about network cost $\zeta(p)$, i.e. the price paid by the network to deliver exchanged chunks. Fig. 16 reports the Cumulative Distribution Function (CDF) of network cost $\zeta(p)$ given a system load ρ equal to 0.75 ($r_s = 1.0$ Mbit/s) . Once more, RTT aware policies significantly reduce index $\zeta(p)$ thanks to their ability to select closer in-neighbors, thus producing a lighter footprint on network resources. Observe that when RTT sensitive policies are adopted, more that 55% of data is exchanged with peers with less than 50 ms RTT. When no location awareness is enabled, this fraction reduces to less than 30%, increasing both network cost and chunk delivery times.

Although RTT-RXC is only the second best in terms of network cost, the difference is marginal, and it is the only policy that can actually combine good performance with low network costs, confirming itself again to be the best choice to adopt.

VIII. Related Work

Many popular commercial applications such as PPLive [23], UUSee [24], PPStream [25], SopCast [26] were proposed in recent years, but no information about their internal implementation has been made available, making any statement about their overlay topology design strategies impossible.

Only a few recent measurement studies suggest that simple random based policies are adopted by SopCast [20], and some slight locality-awareness is implemented in PPlive [27]. Focusing on available literature on purely mesh-based P2P-TV systems, many solutions can be found, but also in this case, to the best of our knowledge, none of them provides *general and detailed* guidelines for the overlay topology design process.

An early solution called *GnuStream* was presented in [28]. Based on *Gnutella* overlay, *GnuStream* implemented a load distribution mechanism where peers were expected to contribute to chunks dissemination in a way proportional to their current capabilities. A more refined solution called *PROMISE* was introduced in [29]. Authors proposed an improved seeder choice based on network tomography techniques; peers were interconnected through *Pastry* overlay topology which implements —as many others P2P substrates like *Chord* [30] or *CAN*— some location awareness based on number of IP hops. *DONet* (or *Coolstreaming*) [2] is a successful P2P-TV system implementation. This design employs a scheduling policy based on chunk rarity and available bandwidth of peers, but its data-driven overlay topology does not exploit any information from underlying network levels. Many new features were introduced in [31] to improve the streaming service and, in particular, authors proposed a new neighbor re-selection heuristic based only on peers up-link bandwidth. In [32], authors showed the design aspects of their application called *AnySee*. Even if partially based on multicast, this hybrid mesh-based system relies on an overlay topology that aims at matching the underlying physical network while pruning slow logical connections. However, no deep investigation about performance of their overlay design strategy is provided. In [33] authors presented a study about some key design issues related to mesh-based P2P-TV systems. They focused on understanding the real limitations of this kind of applications and presented a system based on a directed and randomly generated overlay. Some fundamental improvements were introduced: e.g., the degree of peers' connectivity proportional to their available bandwidth.

Turning our attention on more theoretical studies about the overlay topology formation, in [3] the problem of building an

efficient overlay topology, taking into account both latency and bandwidth, has been formulated as an optimization problem; however, the interactions between overlay topology structure and the chunk distribution process are ignored.

In [34] a theoretical investigation on optimal topologies is formulated, considering latency and peer bandwidth heterogeneity; scaling laws are thus discussed. In [4], a distributed and adaptive algorithm for the optimization of the overlay topology in heterogeneous environments has been proposed, but network latencies are still ignored. Authors of [35] propose a mechanism to build a tree structure on which information is pushed. They show that good topological properties are guaranteed by location awareness schemes. Similar in spirit, but in unstructured systems, we propose in this paper an overlay topology design strategy that, taking into account latency and peer heterogeneity, aims at creating an overlay with good properties and low chunk delivery delays. In highly idealized scenarios, [36] shows with simple stochastic models that overlay topologies with small-world properties are particularly suitable for chunk distribution in P2P-TV systems.

Finally, in [10], authors experimentally compare unstructured systems with multiple-tree based ones, showing that former systems perform better in highly dynamic scenarios as well as in scenarios with bandwidth limitations. This strengthen our choice of exploring topology management policies for mesh-based streaming systems.

IX. Conclusions

P2P-TV systems are extremely complex, and the assessment of their performance through experiments has been rarely undertaken. In particular, the impact of different strategies for the construction and maintenance the overlay topology is of the utmost importance, and remains extremely difficult to study. The work presented in this paper was conceived to fill this gap. Within the framework of the NAPA-WINE project [14], we have developed PeerStreamer, a highly modular and flexible P2P-TV application that allows us select among several different strategies for the construction of the overall topology. In a fully controlled networking environment, we have run a large campaign of experiments measuring the impact of different filtering functions applied to the management of peer neighborhoods. Results show that proper management, based on simple RTT measurements to add peers, coupled with an estimation of the quality of the peer-to-peer relation to drop them, leads to a win-win situation where the performance of the application is improved while the network usage is reduced compared to a classical benchmark with random peer selection. PeerStreamer is released as Open-Source to make results reproducible and to allow further research, but also to build streaming systems and distribute content.

References

[1] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in IPTPS, Ithaca, NY, US, February 2005.

[2] X. Zhang, J. Liu, and T. Yum, "Coolstreaming/donet: A data-driven overlay network for peer-to-peer live media streaming," in *IEEE INFOCOM*, Miami, FL, US, March 2005.

[3] D. Ren, Y. T. H. Li, and S. H. G. Chan, "On Reducing Mesh Delay for Peer-to-Peer Live Streaming," in *IEEE INFOCOM*, Phoenix, AZ, US, April 2008.

[4] 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*, Williamsburg, VA, US, June 2009.

[5] A. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Chunk Distribution in Mesh-Based Large Scale P2P Streaming Systems: a Fluid Approach," *IEEE Trans. on Parallel and Distributed Systems*, vol. 22, no. 3, pp. 451–463, March 2011.

[6] X. Jin and Y.-K. Kwok, "Network aware P2P multimedia streaming: Capacity or locality?" in *IEEE P2P*, Kyoto, JP, August 2011.

[7] Y. Liu, "On the minimum delay peer-to-peer video streaming: how realtime can it be?" in *ACM Multimedia*, Augsburg, DE, September 2007.

[8] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs." in *ACM SIGMETRICS*, Annapolis, MD, US, June 2008.

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

[10] J. Seibert, D. Zage, S. Fahmy, and C. Nita-Rotaru, "Experimental comparison of peer-to-peer streaming overlays: An application perspective," in *IEEE LCN*, Montreal, QC, CA, October 2008.

[11] F. Picconi and L. Massoulié, "Is there a future for mesh-based live video streaming?" in *IEEE P2P*, Aachen, DE, September 2008.

[12] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," *IEEE Trans. on Image Processing*, vol. 13, no. 4, pp. 600–612, April 2004.

[13] 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 *IEEE P2P*, Tarragona, ES, September 2012.

[14] "NAPA-WINE." [Online]. Available: http://www.napa-wine.eu/

[15] 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. Magazine*, vol. 49, June 2011.

[16] 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, IT, October 2010.

[17] 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, no. 18, pp. 2237 – 2244, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366412002332

[18] N. Tölgyesi and M. Jelasity, "Adaptive peer sampling with newscast," in *Euro-Par*, Delft, NL, 2009.

[19] F. Chierichetti, S. Lattanzi, and A. Panconesi, "Rumour spreading and graph conductance," in *ACM-SIAM SODA*, Austin, TX, US, 2010.

[20] I. Bermudez, M. Mellia, and M. Meo, "Passive characterization of SopCast usage in residential isps," in *IEEE P2P*, Kyoto, JP, August 2011.

[21] R. Fortuna, E. Leonardi, M. Mellia, M. Meo, and S. Traverso, "QoE in Pull Based P2P-TV Systems: Overlay Topology Design Tradeoffs," in *IEEE P2P*, Delft, The Netherlands, August 2010. [Online]. Available: http://www.telematica.polito.it/traverso/papers/p2p10.pdf

[22] F. Lehrieder, S. Oechsner, T. Hossfeld, Z. Despotovic, W. Kellerer, and M. Michel, "Can p2p-users benefit from locality-awareness?" in *IEEE P2P*, Delft, NL, August 2010.

[23] "PPLive." [Online]. Available: http://www.pplive.com

[24] "UUSee." [Online]. Available: http://www.uusee.com

[25] "PPStream." [Online]. Available: http://www.ppstream.com

[26] "SOPCast." [Online]. Available: http://www.sopcast.com

[27] L. Vu, I. Gupta, K. Nahrstedt, and J. Liang, "Understanding overlay characteristics of a large-scale peer-to-peer iptv system," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 6, no. 4, pp. 31:1–31:24, Nov. 2010. [Online]. Available: http://doi.acm.org/10.1145/1865106.1865115

[28] X. Jiang, Y. Dong, D. Xu, and B. Bhargava, "Gnustream: a P2P media streaming system prototype," in *IEEE ICME*, Washington, DC, US, 2003.

[29] M. Hefeeda, A. Habib, B. Botev, D. Xu, and B. Bhargava, "Promise: peer-to-peer media streaming using collectcast," in *ACM Multimedia*, New York, NY, US, 2003.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications,"

SIGCOMM Comput. Commun. Rev., vol. 31, pp. 149–160, August 2001.

[31] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *IEEE INFOCOM*, Phoenix, AZ, US, April 2008.

[32] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng, "Anysee: Peer-to-peer live streaming," in *IEEE INFOCOM*, Barcelona, ES, April 2006.

[33] N. Magharei and R. Rejaie, "Prime: Peer-to-peer receiver-driven mesh-based streaming," in *IEEE INFOCOM*, Anchorage , AK, US, May 2007.

[34] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in Peer-to-Peer live multimedia streaming," in *ACM Multimedia*, Santa Barbara, CA, US, October 2006.

[35] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer, "Push-to-Pull Peer-to-Peer Live Streaming," in *Distributed Computing*, ser. Lecture Notes in Computer Science, A. Pelc, Ed. Springer Berlin / Heidelberg, 2007, vol. 4731, pp. 388–402.

[36] J. Chakareski, "Topology construction and resource allocation in p2p live streaming," in *Intelligent Multimedia Communication: Techniques and Applications*, ser. Studies in Computational Intelligence. Springer Berlin / Heidelberg, 2010, vol. 280, pp. 217–251.