Can Peer-to-Peer Live Streaming Systems Coexist with Free Riders?

João F. A. e Oliveira*, Ítalo Cunha*, Eliseu C. Miguel†, Marcus V. M. Rocha‡,
Alex B. Vieira§, Sérgio V. A. Campos*
*Universidade Federal de Minas Gerais †Universidade Federal de Alfenas
‡Assembléia Legislativa de Minas Gerais §Universidade Federal de Juiz de Fora

***Abstract*—Peer-to-Peer live streaming systems help content providers and distributors drastically reduce bandwidth costs by sharing costs among peers. Researchers have dedicated significant effort developing techniques to discourage or exclude uncooperative peers from peer-to-peer systems. However, users are often unable to cooperate, e.g., users using a mobile device with limited, costly bandwidth. We study the impact of uncooperative peers on video discontinuity and latency using PlanetLab. We find that simple mechanisms, like forwarding video data requests to cooperative peers instead of wasting effort sending requests to uncooperative peers, allows peer-to-peer live streaming to serve 50% of uncooperative peers without performance degradation. We argue that denying service to uncooperative peers may not be the best long-term approach; our findings suggest that peer-to-peer live streaming can support uncooperative peers.**

I. Introduction

Peer-to-peer (P2P) networks are used as a content delivery platform to improve the scalability of live streaming systems. Popular live streaming P2P systems, such as TvAnts, UUSee, SopCast, PPLive, and PPStream can have more than eight million users simultaneously watching streams.[1] This number is likely to go up as broadband penetration increases.

P2P live streaming relies on peer cooperation: peers are expected to contribute upload bandwidth redistributing the content they get from one peer to another. Unfortunately, peers may be unwilling or unable to contribute as expected. Uncooperative peers may incur additional load on the video server, compromising system scalability and performance [1].

Significant research effort has been dedicated to the development of mechanisms to discourage uncooperativeness. For example, incentive mechanisms may reward peers with fewer ads or improved stream quality proportional to their contribution [2], [3]. Detection mechanisms try to identify uncooperative peers in order to take corrective measures like removing the peer from the network or reducing their quality of service [4], [5]. Unfortunately, most mechanisms to discourage uncooperativeness incur non-negligible network overhead (e.g., coordination messages), increase system complexity (e.g., accounting of cryptographic receipts [4]), or impose restrictions that limit system performance (e.g., random peer selection [5]).

In this paper, we study the impact of free riders, i.e., extremely uncooperative peers that never upload data, on a real P2P live streaming system. Our system allows control over peer behavior and fine-grained monitoring, normally impossible using proprietary P2P live streaming systems. We perform experiments using PlanetLab [6] to assess the impact of free riders on video playback latency and discontinuity. We also quantify the impact of free riders on server and cooperative peer workloads.

We make three contributions. First, we show that simple modifications, like having uncooperative peers inform their partners that they are unable or unwilling to upload chunks, enables P2P live streaming to serve a large fraction of uncooperative peers without performance degradation (Sec. IV-A). This modification helps because it allows peers to avoid wasting time and bandwidth on chunk requests unlikely to be answered. Second, we show that uncooperative peers that do not inform their partners that they are unable or unwilling to upload chunks, e.g., because of software limitations, client misconfiguration, or malicious intent, cause significant performance degradation (Sec. IV-B).

Third, we propose the Simple Unanswered Request Eliminator (SURE), a modification to the chunk request scheduler that avoids wasting time and bandwidth even if uncooperative peers do not inform their partners (Sec. IV-C). Our experiments show that a P2P live streaming system using SURE can sustain up to 50% of free riders. Compared to a system without free riders, sustaining 50% of free riders increase the median latency by 0.95 seconds and increases average chunk miss rate from 1.09% to 1.59% (still below an acceptable chunk miss rate of 4% [7]). Finally, we show that the additional workload incurred by free riders is evenly balanced among contributing peers and is manageable (Sec. IV-D).

We argue that uncooperative peers occur naturally. For example, it is unrealistic to demand the same contribution from a mobile user with a costly low-bandwidth connection and from a broadband user. Even if a peer is uncooperative for a while, it does not mean that he must be punished; this peer may cooperate normally in a very near future. Instead of dedicating system resources to discourage uncooperativeness [2]–[5], our findings suggest that P2P live streaming systems can tolerate uncooperative peers, increasing the number of viewers, without significant performance degradation.

II. P2P Live Streaming System

One limitation of proprietary commercial systems is that they do not allow control over peer behavior or collection of arbitrary, fine-grained measurements. We implement a P2P live streaming system based on the mesh-pull approach [8], similar to SopCast and PPLive, to address these limitations.

We define a P2P live streaming system as a system with a set of *peers* that collaborate with each other to watch a live media transmission. The *server* is a special peer that encodes the video, splits the video into *chunks* (a chunk can contain multiple frames), and starts the transmission.

[1]Reported on http://www.aplus.pptv.com/aboutus/en.-July2013.

978-1-4799-0521-8/13/$31.00 2013 IEEE

Each peer p has a set of *partner* peers $\mathcal{N}_p$ that p connects to and exchanges video chunks with. In order to join a live streaming channel, a peer p registers itself at a centralized *bootstrap* server, which returns to p a subset of all peers currently active in the system as potential partners. Peer p selects peers from this subset and tries to establish partnerships with them. Successfully established partnerships determine $\mathcal{N}_p$. When p detects that one of its partners, say p', has been silent for longer than a predefined time period, p removes p' from $\mathcal{N}_p$. In this case, peer p may also contact the bootstrap server to obtain a new list of potential partners to replace the old partnership. In this work, we set the default number of partners (also called the *neighborhood size*) to 20, similar to other commercial P2P live streaming systems [9].

Each peer has a local buffer to store its video chunks. Periodically, peers exchange *buffer maps* with their partners to inform them what chunks they have available. Each peer periodically checks which chunks it needs, identifies which partners can provide missing chunks, and sends chunks requests accordingly. Peers may schedule requests depending on chunk availability, e.g., rarest chunk first, or they may schedule depending on playback time, e.g., earliest deadline first. The rarest first policy tries to replicate a chunk as soon as possible while the earliest deadline policy tries to make playback smoother. In this work, we schedule chunk requests using the earliest deadline first policy. If multiple partners have a chunk, the scheduler chooses one of them at random. We do not limit the number of simultaneous (pending) requests. We limit the number of request retries to six to control each peer's opportunities to download a chunk and make download opportunities independent of buffer size. A peer considers a request has timed out if it is not answered within 500 milliseconds. Finally, peers immediately serve the requests they receive in order of arrival.

Peers send monitoring reports to the bootstrap server every ten seconds. Reports currently include the number of chunks generated (only reported by the video server), sent, received, and the ones that missed their playback deadline; the number of requests sent, answered, and retried; the average forwarding path length, retry count, and time of arrival of received chunks; neighborhood size; and the number of duplicate chunks received. We use peer reports to compute the performance metrics we evaluate: chunk latency and the chunk playback deadline miss rate.

III. Experimental Method

Our evaluation relies on real experiments that we have conducted on PlanetLab. We configured video and bootstrap servers in our university's network and used about 450 PlanetLab nodes as peers. The video server streams a 420 kbps video (about 40 chunks per second). PlanetLab nodes run multiple simultaneous experiments, so each peer has variable CPU and bandwidth constraints. Thus, we do not impose any additional resource constraints on peers.

All peers join the live transmission during an initial period of 60 seconds, with joining times chosen randomly following an uniform distribution. Each experiment lasts 8 minutes, and we discard data from the first and last 90 seconds (i.e., the warm up and cool down periods).

We consider different scenarios, varying the behavior and fraction of free riders. For each type of free rider behavior, we vary the fraction of free riders from 0% to 95% in steps of 5%. For each configuration, we run at least ten experiments and report results over all experiments. To maximize the impact of free riding, we do not use any mechanism to choose or abandon partnerships (e.g., reputation systems). In other words, peers do not drop or punish uncooperative partners.

We define two types of free rider, namely *conscious* and *oblivious*. Conscious free riders inform their partners that they are unwilling or unable to upload data. This behavior may be coded in the software or chosen by users. In our system, conscious free riders request chunks as normal but always advertise empty buffer maps. As a consequence, no peer ever wastes time and bandwidth sending requests to conscious free riders. Oblivious free riders do not inform their partners that they are unwilling or unable to upload data. This behavior may happen if the software does not make provisions for free riders, if the user misconfigures the client, or due to malicious intent. In our system, oblivious free riders request chunks as normal and advertise buffer maps with the chunks they have, but never upload data (i.e., never answer requests). Oblivious free riders may receive requests and degrade system performance, as their partners will have to retransmit chunk requests after waiting for answers that never arrive.

We quantify the impact of uncooperativeness on P2P live streaming systems using chunk latency and chunk playback deadline miss rate. Chunk latency, also called diffusion latency, is the delay between the creation of chunk (at the video server) and its reception by a peer. We define the playback deadline miss rate as the fraction of chunks that are not received before their playback deadline. High latency is undesirable as peers will play outdated content (e.g., a neighbor cheering a goal that you will watch ten seconds from now). Missed chunks cause flickering or interruption, specially when many chunks are lost in sequence.

IV. Evaluation

In this section we evaluate the impact of free riders on P2P live streaming performance. We show that conscious free riders have minor impact on performance, but that oblivious free riders seriously degrade performance. We then introduce SURE, the Simple Unanswered Request Eliminator, to mitigate the impact of oblivious free riders. Finally, we show that the increased workload caused by free riders is evenly balanced among cooperative peers and is manageable.

Unless stated otherwise, we aggregate results from all ten experiments performed for each scenario and the coefficient of variation of reported averages are below 0.5.

A. Conscious Free Riders

Fig. 1 shows the distribution of average chunk latency over all peers in the channel. We plot multiple curves varying the fraction of conscious free riders. The average latency is representative of a peer's chunk latencies: the standard deviation of chunk latencies is less than 1 second for 93% of peers. Fig. 1 shows that latency stays stable if the number of conscious free riders is less than 10%. As the fraction of free riders increases, the fraction of partners of a peer

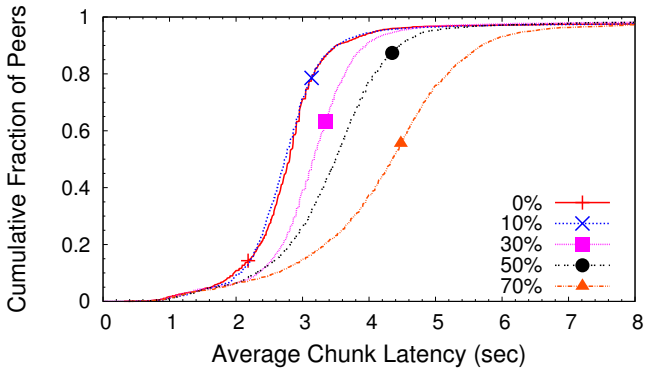

Figure 1: Distribution of average chunk latency for varying fractions of conscious free riders.

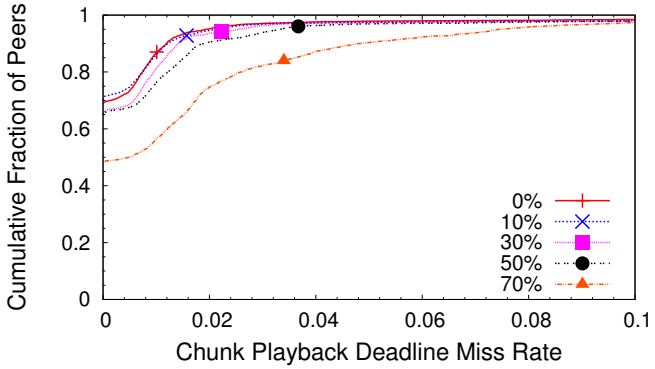

Figure 2: Distribution of chunk playback deadline miss rate for varying fractions of conscious free riders.

willing to contribute decreases. This shortage of contributing partners causes chunk forwarding paths to become longer. The median chunk forwarding path length is 3.5 for 10% of free riders, but increases to 4.1 for 50% of free riders. Longer forwarding paths increase latency since there are delays associated with buffer map advertisements, chunk requests, and the data transfer itself.

Fig. 2 shows the distribution of chunk playback deadline miss rate over all peers. Again, we plot various curves varying the fraction of conscious free riders. We note that some chunks may miss their playback deadline for factors other than uncooperativeness. For instance, our results include PlanetLab nodes that may be overloaded, lacking enough network bandwidth and CPU to download and process chunks.

The chunk miss rate is qualitatively similar for fractions of free riders below 50%. With less than 50% of free riders, 65% of peers receive all chunks before their playback deadline and 96% of peers experience chunk miss rates lower than 4% (acceptable video quality [7]). Chunk miss rate increases significantly with 70% of free riders. Peers have few partners that can provide chunks and they may fail to receive a chunk before its playback deadline. If a content provider intends to sustain a system with 70% (or more) of free riders, it may need a hybrid architecture with well-provisioned support peers to cover the missing resources.

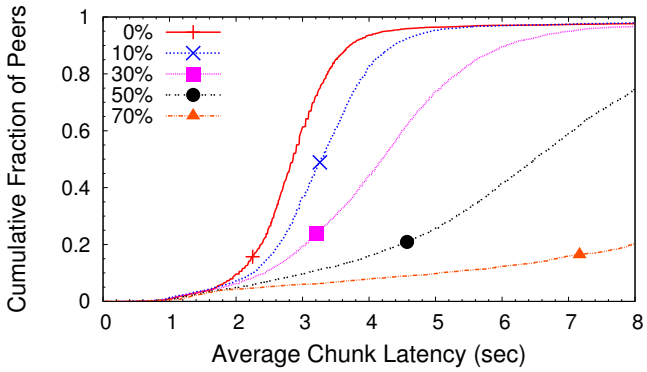

Figure 3: Distribution of chunk latency for various fractions of oblivious free riders.

B. Oblivious Free Riders

When a peer requests a chunk from an oblivious free rider, his request will time out and he will need to retry later. In particular, the chance that retries are (repeatedly) requested to an oblivious free rider is proportional to the fraction of free riders in the network.

We compare the number of request retries in scenarios with conscious and oblivious free riders. Retries are rarely needed when free riders are conscious (e.g., when a peer removes a chunk from its local buffer while a request is in transit). Thus, peers usually make a single request per chunk, with a median of 1.007 tries per hop in the chunk forwarding path. When free riders are oblivious, the number of retries to obtain a chunk increases significantly. For instance, with 50% of oblivious free riders, the median number of tries for each hop on the chunk forwarding path is 1.4.

Consecutive retries waste bandwidth and increase average chunk latency. Moreover, consecutive retries may increase the chunk miss rate if no retry succeeds before the chunk playback deadline. Fig. 3 shows the distribution of average chunk latency over all peers in the channel. Even 10% of oblivious free riders increase the median of the distribution of average chunk latency to 3.28 seconds, a 15% increase compared to the scenario without free riders. This is significantly worse than having 10% of conscious free riders (Fig. 1), which have almost no impact on latency. System performance gets even worse when the fraction of oblivious free riders increases. The median average chunk latency is 6.49 seconds if the channel has 50% of free riders, a 127% increase compared to the scenario without free riders. Similar degradation happens for the chunk miss rate (not shown).

C. Simple Unanswered Request Eliminator (SURE)

Chunk request retries are the major difference between scenarios with conscious and oblivious free riders. If we could avoid chunk requests to oblivious free riders in the first place, we would limit their negative impact on system performance. We introduce and evaluate SURE, a modification to our system's request scheduler that avoids excessive retries caused by oblivious free riders. Our goal is to show that even simple solutions can significantly reduce the impact of uncooperativeness on P2P live streaming systems.

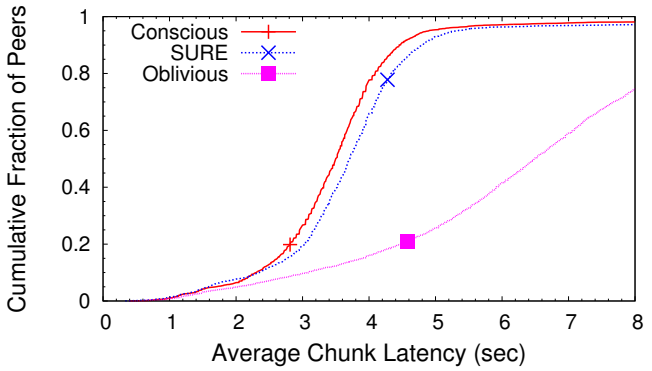

Figure 4: Distribution of average chunk latency for different free rider behaviors and SURE (50% of free riders).

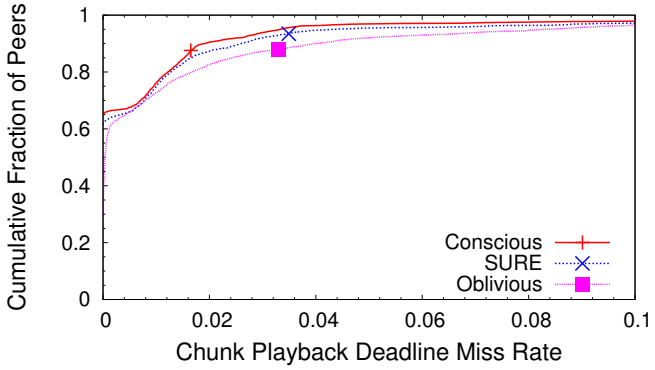

Figure 5: Distribution of chunk playback deadline miss rate for different free rider behaviors and SURE (50% of free riders).

SURE maintains a counter of pending requests for each partner. Whenever a peer sends a chunk request to a partner, it increments that partner's pending requests counter. Whenever a peer receives a chunk from a partner, it decrements that partner's counter. The idea is that counters for oblivious free riders will increase rapidly, while counters for contributing peers will remain low. When sending a chunk request, peers choose the partner with the smallest counter among partners with that chunk. If multiple partners have the same amount of pending requests, SURE picks one at random.

Fig. 4 shows the distribution of average chunk latency over all peers for 50% of conscious and oblivious free riders (curves "conscious" and "oblivious" in Fig. 4 are the same as "50%" in Figs. 1 and 3). We also plot the chunk latency for a scenario with 50% of oblivious free riders when using SURE. SURE reduces the number of retries. The chunk latency is almost equivalent to the baseline scenario of conscious free riders. Results for different fractions of free riders lower than 50% are qualitatively similar.

Similarly, Fig. 5 shows the distribution of chunk playback deadline miss rate for 50% of conscious and oblivious free riders; as well as the miss rate for oblivious free riders when using SURE. Again, SURE reduces the chunk miss rate to levels equivalent to those of the baseline scenario with conscious free riders. In particular, SURE reduces by half the fraction of peers with chunk miss rate higher than 4%.

Table I: Impact of Free Riders on Workload Distribution

Free Riders	Uncooperative $0 < C \leq 1$		Cooperative $1 < C \leq 5$		Altruistic $C > 5$	
%	%	$E[C]$	%	$E[C]$	%	$E[C]$
0%	69.9%	0.34	26.9%	1.86	3.2%	8.15
10%	59.2%	0.35	27.3%	1.86	3.5%	8.02
30%	40.9%	0.38	24.9%	1.93	4.2%	8.66
50%	22.6%	0.42	22.0%	2.07	5.4%	8.31
70%	7.3%	0.43	16.1%	2.41	6.5%	8.86
75%	5.5%	0.41	12.9%	2.61	6.6%	9.71
80%	3.6%	0.31	8.3%	2.83	8.1%	9.28
85%	2.8%	0.26	4.2%	3.39	8.1%	10.54

SURE works because it identifies uncooperative peers with few interactions. When a peer joins the channel, all its partners have zeroed pending requests counters and it may send requests to uncooperative partners. However, uncooperative partners' pending requests counters increase quickly and they are avoided until the end of the partnership. Another advantage of SURE is that it balances the load among peers. Consider that two partners have many desirable chunks and the same value on their pending requests counters. When SURE issues a request to one of the partners and increments its pending requests counter, it will prefer the other partner for the next request as it will have a smaller counter. We present SURE to show that even simple solutions allow P2P streaming systems to mitigate most of the impact of uncooperative peers on system performance. We leave the evaluation of recovery mechanisms (e.g., slowly decrementing counters over time, optimistic unchoke) as future work. Recovery mechanisms could improve performance in long videos or in scenarios where peers change behavior. Finally, we note that other alternatives to eliminate unanswered requests are possible.

D. Network Load Induced by Uncooperative Peers

A natural question is what happens to the workload distribution when uncooperative peers do not contribute to the system's aggregate upload capacity; there must be nodes bearing the workload. Tab. I shows what happens to the workload distribution when the fraction of free riders increases. We categorize peers using their *cooperation level* C, i.e., the ratio of a peer's average upload rate relative to the video bitrate throughout the experiment. We categorize peers as free riders if $C = 0$, uncooperative if $0 < C \leq 1$, cooperative if $1 < C \leq 5$, and altruistic if $C > 5$. Tab. I presents the fraction of peers in each category and their average cooperation level. We show results from the experiments with oblivious free riders and SURE, but results with conscious free riders are quantitatively similar.

Even when there are no free riders in the system, most peers are classified as uncooperative. A large fraction of network load is carried by a small number of cooperative and altruistic peers. This unbalanced load distribution happens because (i) the P2P system does not use any load balancing scheme and because (ii) peers closer to the server receive chunks early and have more time to redistribute chunks than peers far from the server. Uncooperative peers occur due to intrinsic protocol mechanisms, as also observed in SopCast [10].

As the fraction of free riders increases, workload on cooperative peers increases and they shift to the altruistic category. However, the average workload per peer in the cooperative and altruistic category remains stable close to 2 and 8 stream rates, respectively, if there are less than 70% of free riders.

Tab. I shows that 70% of peers are uncooperative even when there are no free riders (first line). However, previously cooperative peers are turned into free riders in scenarios with more than 70% of free riders, resulting in significant workload increase on the remaining cooperative and altruistic peers. After this turning point, system behavior converges quickly to a client-server model (like Youtube), where most peers contribute nothing and a few well-provisioned peers sustain all the workload. Finally, we note that although 70% of free riders is the upper limit before system collapse, performance degradation starts after 50% of free riders, as discussed in Secs. IV-A, B, and C.

V. Related Work

Impact of free riders. One of the first studies about free riding identified that 63% of Gnutella peers never answered a file search query [1], meaning that few peers were responsible for supplying most of the data and maintaining the network. Since then, several works have studied the impact of free riding on P2P file sharing and streaming systems (see the work of Karakaya et al. [11] and Moltchanov [12] and references within). More similar to our work, some studies on P2P file sharing networks have shown that free riders are not as detrimental as originally believed [13], [14]. For example, Meulpouder et al. [14] found that in closed communities that enforce high cooperation levels, peers compete to upload data but may lack opportunities to do so. Thus, even peers willing to cooperate might not survive the upload competition, being punished or removed from the file sharing community.

Free riding mitigation. Researchers have studied two main approaches to the problem of uncooperativeness in live streaming: first, the development of incentive mechanisms to encourage cooperation, and second, the development of techniques to identify and punish uncooperative peers. A well-known incentive mechanism used in P2P file sharing is BitTorrent's tit-for-tat [15]. Unfortunately, tit-for-tat is inadequate for live streaming. Tit-for-tat forces balanced pairwise data exchanges, which are too restrictive to enable live streaming [4]. Some works have attempted to use tit-for-tat as a feature, where uncooperative peers experience degraded quality of service, even at overprovisioned scenarios [16].

Contracts [4] is a mechanism to identify uncooperative peers that uses system resources (bandwidth and CPU) to audit peer cooperation using cryptographic receipts of chunk transfers. LiFTinG [5] identifies uncooperative and malicious peers based on their partnerships. Unfortunately, LiFTinG requires that peers make partnerships at random, which may be suboptimal, reducing system performance. Our argument in this paper is that P2P live streaming systems can sustain a large fraction of uncooperative peers without performance degradation. We believe that, unless a system is under attack, these identification mechanisms may be unnecessary and even waste system resources.

VI. Conclusion

P2P live streaming is an important application. The impact of uncooperative peers on system scalability and performance has been a major concern. In this paper, we identify that request retries caused by uncooperative peers' inability or unwillingness to answer video chunk requests waste bandwidth and time, degrading performance. We also show that P2P live streaming can deal satisfactorily with uncooperative peers by implementing measures that avoid sending requests to uncooperative peers. For instance, our PlanetLab-based experiments show that our P2P live streaming system can sustain 50% of free riders with negligible performance degradation. We also find that the workload incurred by free riders is evenly balanced among contributing peers and is manageable.

We argue that systems should tolerate free riders. Instead of dedicating resources to discourage or remove uncooperative peers from the system, researchers should focus on designing mechanisms to increase system scalability and performance in face of free riders, ultimately increasing the number of viewers.

References

[1] E. Adar and B. Huberman, "Free Riding on Gnutella," *First Monday*, vol. 5, no. 10-2, 2000.

[2] B. Wang, A. Chow, and L. Golubchik, "P2P Streaming: Use of Advertisements as Incentives," in *ACM MMSys*, 2012.

[3] Z. Liu, Y. Shen, K. Ross, S. Panwar, and Y. Wang, "LayerP2P: Using Layered Video Chunks in P2P Live Streaming," *IEEE Transactions on Multimedia*, vol. 11, no. 7, pp. 1340–1352, 2009.

[4] M. Piatek, A. Krishnamurthy, A. Venkataramani, R. Yang, D. Zhang, and A. Jaffe, "Contracts: Practical Contribution Incentives For P2P Live Streaming," in *USENIX NSDI*, 2010.

[5] R. Guerraoui, K. Huguenin, A.-M. Kermarrec, M. Monod, and S. Prusty, "LiFTinG: Lightweight Freerider-Tracking in Gossip," in *ACM/IFIP/USENIX International Conference on Middleware*, 2010.

[6] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir, "Experiences Building PlanetLab," in *USENIX OSDI*, 2006.

[7] 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*, 2012.

[8] X. Hei, Y. Liu, and K. Ross, "IPTV Over P2P Streaming Networks: The Mesh-Pull Approach," *IEEE Communications Magazine*, vol. 46, no. 2, pp. 86–92, 2008.

[9] A. Borges, P. Gomes, J. Nacif, R. Mantini, J. M. Almeida, and S. Campos, "Characterizing SopCast Client Behavior," *IEEE Computer Communications*, vol. 35, no. 8, pp. 1004–1016, 2012.

[10] G. D. Gonçalves, A. Guimarães, A. B. Vieira, I. Cunha, and J. M. Almeida, "Using Centrality Metrics to Predict Peer Cooperation in Live Streaming Applications," in *IFIP Networking*, 2012.

[11] M. Karakaya, I. Korpeoglu, and O. Ulusoy, "Free Riding in Peer-to-Peer Networks," *IEEE Internet Computing*, vol. 13, no. 2, pp. 92–98, 2009.

[12] D. Moltchanov, "Service Quality in P2P Streaming Systems," *Computer Science Review*, vol. 5, no. 4, pp. 319–340, 2011.

[13] R. Krishnan, M. Smith, Z. Tang, and R. Telang, "The Impact of Free-Riding on Peer-to-Peer Networks," in *Annual Hawaii International Conference on System Sciences*, 2004.

[14] M. Meulpolder, L. Meester, and D. Epema, "The Problem of Upload Competition in Peer-to-Peer Systems With Incentive Mechanisms," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 7, pp. 899–917, 2012.

[15] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[16] T. Locher, R. Meier, R. Wattenhofer, and S. Schmid, "Robust live media streaming in swarms," in *NOSSDAV*. ACM, 2009, pp. 121–126.