POLITECNICO DI TORINO
Repository ISTITUZIONALE

P2P-TV systems under adverse network conditions: a measurement study

Original
P2P-TV systems under adverse network conditions: a measurement study / Alessandria, E; Gallo, M; Leonardi, Emilio; Mellia, Marco; Meo, Michela. - (2009), pp. 100-108. (Intervento presentato al convegno IEEE Infocom 2009 tenutosi a Rio de Janeiro nel 21 April 2009) [10.1109/INFCOM.2009.5061911].

Availability:
This version is available at: 11583/1906915 since:

Publisher:
IEEE

Published
DOI:10.1109/INFCOM.2009.5061911

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)

19 March 2024

P2P-TV Systems under Adverse Network Conditions: a Measurement Study

Eugenio Alessandria, Massimo Gallo, Emilio Leonardi, Marco Mellia, Michela Meo
Politecnico di Torino
Email: lastname@tlc.polito.it

***Abstract*—In this paper we define a simple experimental set-up to analyze the behavior of commercial P2P-TV applications under adverse network conditions. Our goal is to reveal the ability of different P2P-TV applications to adapt to dynamically changing conditions, such as delay, loss and available capacity, e.g., checking whether such systems implement some form of congestion control. We apply our methodology to four popular commercial P2P-TV applications: PPLive, SOPCast, TVants and TVUPlayer. Our results show that all the considered applications are in general capable to cope with packet losses and to react to congestion arising in the network core. Indeed, all applications keep trying to download data by avoiding bad paths and carefully selecting good peers. However, when the bottleneck affects all peers, e.g., it is at the access link, their behavior results rather aggressive, and potentially harmful for both other applications and the network.**

I. Introduction and Motivations

A new class of peer-to-peer (P2P) systems providing real-time video streaming over the Internet is fast emerging and gaining popularity. Several commercial P2P streaming systems such as PPLive [1], SOPCast [2], TVants [3] and TVUPlayer[4], just to mention the most popular ones, are already available on Internet. This first generation of P2P-TV systems offers moderate-quality P2P video streaming (200–400 kbit/s); a next generation high-quality P2P streaming (1–10 Mbit/s) is just beyond the corner, as commercial P2P video applications, such as Joost [5], Babelgum [6], Zattoo [7] are at an advanced stage of beta-testing and are likely to be fully launched in the next few months.

P2P-TV systems may contribute to revolution the broadcast TV paradigm allowing ubiquitous access to a practically unlimited number of channels. This represents an important step forward in the direction of an Anything/Anyone/Anywhere/Anytime (A4) ubiquitous communication paradigm of future Internet applications [8].

From a technical point of view, the adoption of a P2P paradigm reduces the network costs, pushing complexity from the network to the users, while helping to relieve the bandwidth cost burden at the server. Although from the users' as well as from the server points of view this class of P2P applications has useful and interesting characteristics, from the network operators' point of view serious concerns exist about the capability of the Internet to support large scale P2P-TV systems (mainly due to the potential high bandwidth requirements, large number of involved users, and the intrinsic inelasticity of transported traffic). These concerns are confirmed by some news reports, see for instance [9].

It seems, therefore, rather urgent to have a better understanding of the potential impacts that these applications may entail on the underlying transport network. Since the most widely deployed commercial systems cited above follow a closed and proprietary design, only an experimental behavioral (black-box) characterization of traffic injected by such systems is in general possible; we emphasize, indeed, that approaches requiring to partially reverse the engine of P2P-TV systems are viable, at larger cost, only in a few cases. Also, in order to develop new architectures and algorithms that improve the "network friendliness" of such applications [10], [11], it is necessary to understand how current applications react to different network conditions and scenarios. Do they implement any congestion control algorithm? How do they react to packet drop? What is the impact of increased end-to-end delay?

In this paper we accomplish a first step, by defining a methodology that helps in answering those questions. We propose simple test-bed experiments to assess how these applications react to different network conditions, like available bandwidth, loss probability, delay; both network load, and user perceived quality of service, should then be measured. Applying our methodology, we test and compare four popular P2P-TV applications, namely PPLive, SOPCast, TVants and TVUPlayer. All selected applications adopt the "mesh-based" approach [8], in which peers form a generic overlay topology used to exchange chunks of data among neighboring peers.

Results show that all applications are effective in trying to overcome network impairment. For example, all applications avoid impaired paths by carefully selecting peers to download from. However, when the bottleneck affects all paths, e.g., in case they access link is congested, the aggressively download data trying to receive the video stream. This leads to overload factors larger than two in some scenarios. While the behavior of P2P-TV guarantees to offer good end-user service even in presence of adverse network conditions, it poses a problem when considering the network and application friendliness of P2P-TV applications.

We discover then that all applications implement a memory based algorithm that tracks good and bad neighbor peers,

This work was funded by the European Commission under the 7th Framework Programme Strep Project "NAPA-WINE" (Network Aware Peer-to-Peer Application under WIse Network.)

while no change is observed in the mechanisms to create the neighbors set.

The paper is organized as follows: we start by summarizing the related work in Sec. II. Then, the measurement setup and methodology are defined in Sec. III. Sec.IV presents then results in which network impairment affects all peer, while Sec. V investigate scenarios in which "good" and "bad" peers are present so to investigate the capability of the applications to correctly handle them. Finally, Sec. VI summerizes our findings.

II. Related Work

To the best of our knowledge, this is the first experimental work on P2P-TV systems exploring how such systems react to different network conditions. In a previous paper, we performed a similar characterization considering Skype [12], in which the focus was on the voice traffic sent/received by a Skype client.

Considering more general experimental results about P2P-TV systems, the research community has given a lot of attention to understand application internals [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23].

A few works [13], [14], [15], relying on the implementation of an active crawler, focus on a single system. These approaches face the daunting task of partially reversing the engine of the P2P-TV system under analysis. As a consequence, this methodology is limited by the ability to break closed and proprietary systems, and we believe that they can be hardly extended to characterize all the possible P2P-TV applications. In particular, [13] investigates PPLive, whereas [14] focuses on the commercial re-engineer of Coolstreaming, and [15] considers UUSee. All these papers mainly provide a big picture of the considered P2P-TV, focusing on metrics such as the number of users in the systems, their geographical distribution, the session duration of users, the distribution of packet size. None of the above mentioned paper considers the particular aspects we are interested into, i.e., the way the system reacts to network conditions.

Other works, such as [16], [17], instead, study specific aspects of a P2P streaming system. For instance, [16] gives some preliminary results on the node degrees of popular versus unpopular channels in PPLive. Authors in [17] investigate the stability of nodes in P2P live video streaming system, considering again PPLive, and devising schemes to identify the most stable nodes in the system.

Quality of service is of concern in [18], [19]. Authors in [18] exploit an analysis of PPLive buffer maps, collected through protocol reverse engineering, to infer QoS metrics such as network-wide playback continuity, startup latency, playback lags among peers, and chunk propagation timing. Authors in [19] focus on similar metrics but exploit logs made available from an (unspecified) commercial P2P streaming system.

Authors in [20] analyze and compare PPLive and SOPCast investigating the time evolution of different metrics, like transmitted/received bytes, number of parents and children, etc. Paper [21], on the contrary, presents a comparative evaluation of four commercial systems (namely PPLive, PPStream, SOPCast and TVAnts) and compares these systems showing flow-level scatter plots of mean packet size versus flow duration and data rate of the top-10 contributors versus the overall download rate. In [22] PPLive, SOPCast and TVAnts systems are analyzed. A systematical exploration of the mechanism driving the peer-selection in the different systems is performed. At last, in [23] a simple experimental analysis of PPLive and Joost is presented to evaluate the characteristics of both data distribution and signaling process for the overlay network discovery and maintenance.

III. Methodology

The aim of this work is to study how P2P-TV applications react to different network scenarios. Given that all successful P2P-TV applications follow a proprietary and closed design, we have to follow a "black-box" approach. We therefore setup a testbed, in which a client running the Application Under Test (AUT) is connected to a Linux router, which is in turn connected to the Internet via our Fast-Ethernet based campus LAN. The router itself is then used to enforce particular network conditions. In particular, we used the Linux Network Emulation functionality `netem` coupled with the Token Bucket Filter `TBF`. This allows us to emulate the properties of wide area networks, controlling available bandwidth, delay, loss, duplication and re-ordering of packets routed through the router.

Note that, since we run real on-field experiments, our control on the experimental set-up is limited to the interface in object only. This implies that the global network conditions are unknown and that possible effects due to congestion, loss, delay inside the Internet are superposed to the effects "artificially" introduced at the router under our control.

Two packet level traces are then collected at the router: the first one logs all packets sent/received by the network interface that connects the router to the Internet; the second one logs all packets sent/received by the network interface that connects the PC running the AUT. Packet level traces are then post-processed to obtain the desired measurements. In this paper, we report results considering only the traffic *received* by the Internet interface, reporting in particular the *average received bit-rate* $r(t)$ evaluated at the application layer, i.e., neglecting transport, network and data-link overheads. The number of peers that sent packets to the AUT during a time interval $n(t)$, i.e. the *number of active peers*, is reported as well.

Finally, the PC running the AUT is used to capture the video stream that is received and to dump it on a file by means of a video grabber utility. To evaluate the video quality of the received stream, we cannot apply any standard technique, since they all rely on the comparison of the received and original video (being impossible to get the latter one). In addition, all selected applications utilize a 378 kbps stream encoded using the Microsoft VC-1 encoder in all tests; a bit-rate of 450-500 kbps is received by the AUT, so that 100-150 kbps of additional overhead is required by the applications to successfully deliver the stream (not including transport, network

and data-link headers). Since the codec relies on proprietary design, it is difficult to evaluate the quality of the received stream. We are therefore forced to estimate the stream quality by simply counting the number of errors a decoder has to manage when decoding the stream. In particular, we decoded each file using `ffmpeg` utility which reports a detailed list of corrupted video I-frames. Those are major impairment that will affect the video quality for several frames, i.e., up to when a good I-frame is received (usually several seconds). Similarly, the audio stream decoding errors are evaluated as reported by `ffmpeg`. In the following, we report therefore the number of corrupted I-frames and audio blocks as quality index. While this allows only a qualitative evaluation of the stream quality, it permits us to fairly compare different applications.

A. Scenario definition

The parameters we consider in this paper are the following:

- c: Capacity limit
- l: Packet loss
- d: Delay

The array $L = \{c(t), l(t), d(t)\}$ specifies the state of the controlled link during the experiment - we restrict to the cases in which only one of the three above parameters is evolving with time and we denote with $p(\cdot)$ its *profile* over time.

As profile $p(\cdot)$ we select a step function, with initial value p_0, increments I, and step duration ΔT, so that

$$p(p_0, I, \Delta T) = p_0 + I \sum_{n=0} H(t - n\Delta T) \tag{1}$$

in which $H(t)$ is the Heaviside step function

$$H(t) = \begin{cases} 0 & t < 0 \\ 1 & t \geq 0 \end{cases} \tag{2}$$

If I is positive, $p(\cdot)$ corresponds to an *increasing* profile; negative values of I, on the contrary, generate *decreasing* profiles. A null increment, $I = 0$, finally, leads to a *constant* profile.

The impairment defined by a scenario can affect all sent/received packets, so that *global* impairment is imposed, or only a subset of sent/received packets, so that *per peer* impairment is imposed; for example the scenario can affect a single peer, a subnet, an Autonomous System, or any generic subset of IP addresses.

B. Considered general setup

We performed several experiments considering different scenarios and profiles, during various time periods and with clients tuned on different TV channels. We collected a total of more than 300 hours of experiments. In this paper, we report a subset of the most representative experiments. In particular, we consider only scenarios in which the download link is controlled, while the upload link capacity is limited to $c_{up}(t) = 200$ kbps. This indeed allows us to evaluate the application behavior when the peer has not enough capacity to act as an "amplifier", i.e., to serve many peers; this is the typical condition of ADSL users.

In the following, we show results considering scenarios in which one parameter at a time varies.

IV. Global impairment

A. Effect of available capacity

In the first set of experiments we report, the download available capacity $c(t)$ is imposed. Results are shown in Fig. 1 and are organized in the following way. The two largest plots in the left part of the figure report the bit-rate $r(t)$ versus time for the four considered applications, for profiles with either decreasing or increasing capacity limit (on the top and bottom plots respectively). The 8 small plots on the right part of the same figure depict the number of corrupted audio and video frames for the same experiments; each plot refers to a specific application.

Let us start by considering the decreasing profile. Every $\Delta T = 5$ minutes, the available bandwidth is decreased by a $I = -50$ kbps, starting from an initial value of $c_0 = 800$ kbps. The average bit-rate evaluated using 60 seconds time intervals is reported for all applications on left top plot of the figure. The experiment lasted 1 hour, after which the available capacity was set back to 800 kbps. All applications have similar behavior: the bit-rate remains basically constant for all the time the available capacity is larger than the data rate, $c(t) > r(t)$. When the capacity bottleneck kicks in, all the applications react by increasing the download data rate. Consider, for example, TVAnts, which exhibits the largest value of the bit-rate. The normal data rate is about 600 kbps; when the capacity limit reaches 650 kbps, the receiver starts suffering the bottleneck (due to traffic burstiness), and it reacts by asking other peers to send more traffic; the download rate becomes larger than 800 kbps. Interestingly, as the capacity limits $c(t)$ decreases, the received rate decreases also, being always about twice the available capacity, i.e., the offered load to the congested link is about 2, $r(t)/c(t) \simeq 2$. The other applications show similar behavior, with smaller values of the offered load in congested conditions; in particular, TVUPlayer exhibits the smallest overload factor $r(t)/c(t) \simeq 1.3$.

From these results, we can conclude that the considered applications do not correctly identify the globally limited available bandwidth. They assume that congestion is not affecting all peers, e.g., as it happens when the bottleneck is at the access network, but, rather, in some specific paths only. Reduced performance is then faced by trying to retrieve the video stream from other peers in possibly different network locations (not affected by congestion). This is further investigated in Sec. V. By acting this way, if the application itself causes congestion, it actually ends up worsening the situation and further increasing congestion.

Looking at the last 10 minutes of the experiment, when the capacity returns to high values, an unexpected, strange behavior is observed. Indeed, since $c(t)$ is larger then the normally required capacity, $r(t)$ should take again the typical values that can be observed when no bottleneck is present. While this is true for PPLive and SopCast, both TVAnts and TVUPlayer keep receiving at a rate which is about twice as

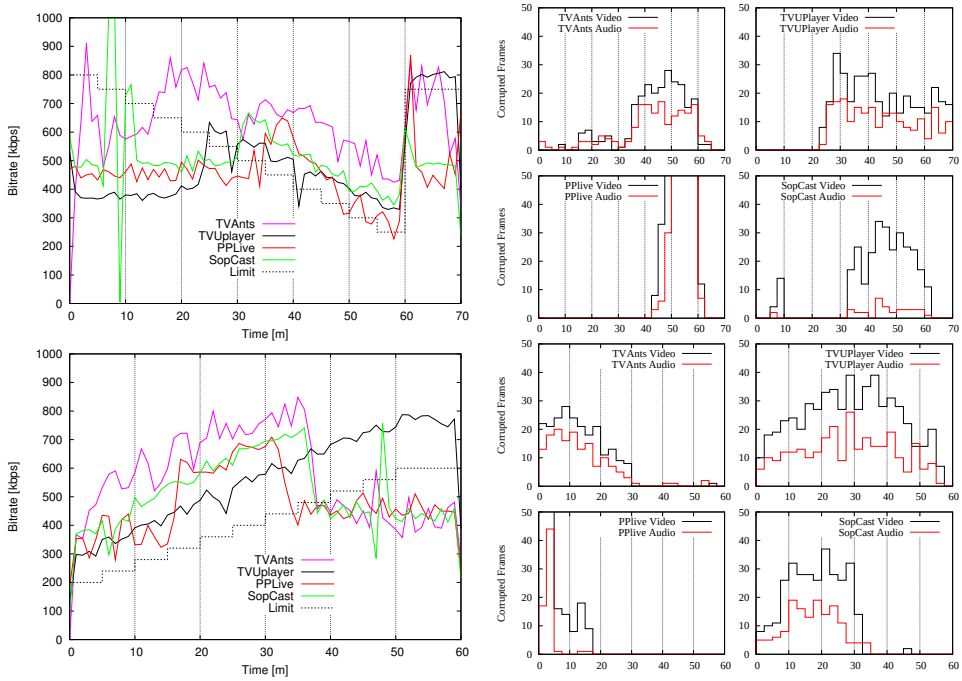


Fig. 1. Left plots: received bit-rate for decreasing (top) and increasing (bottom) available bandwidth. Right plots: percentage of corrupted audio/video frames, one plot for each application.

large as the normal receiver rate. This maintains the bottleneck offered load higher than 1, so that audio/video impairment can be noted up to the end of the experiment. Indeed, looking at the number of corrupted frames reported in the top right part of Fig. 1, audio/video impairment starts to show up as soon as the bottleneck kicks in, and it does not always disappear when the bottleneck capacity is set back to 800 kbps. TVAnts shows the longest period during which corrupted frames are observed, while, PPLive can cope with downlink capacity as small as 400 kbps without any audio/video error.

Results for the case of an increasing capacity profile are reported in bottom part of Fig. 1. The AUT is started now in scarce bandwidth conditions ($b_0 = 200$ kbps), and $I = 40$ kbps increments are applied every $\Delta T = 5$ minutes. All applications react to the adverse condition by trying to download much more data than the available capacity; also in this case $r(t)/c(t)$ is between 1.3 (for TVUPlayer) and 2 (for TVAnts). Only when the available capacity is large enough to sustain the minimal download rate, all applications but TVUPlayer decrease $r(t)$ to their typical values. This is reflected by the disappearance of audio/video impairment, as shown by the right plots.

We can conclude that P2P-TV applications do not correctly perform congestion control, in scenarios in which peer access links get congested. They all try and react to limited access capacity by increasing the redundancy (by FEC or ARQ mechanisms) and, thus, the download rate. This may be potentially harmful for both the network and other applications sharing the congested link.

Note that a single congested link may also be present when the unique peering link between a stub ISP and the rest of the Internet gets congested. If this happens, P2P-TV applications may react as in the previously presented cases, causing further network problems and congestion.

B. *Effect of loss probability*

The second set of results we report aims at investigating the impact of loss probability on the AUT. Organized in a similar way as the previous figure, Fig. 2 shows the receiver rate for increasing (top plots) and decreasing (bottom plot) packet loss probability profiles. The right y-axis of the bit-rate plots reports the percentage of losses, $l(t)$ that varies in time steps of $\Delta T = 5$ minutes, with loss increment $I = 5\%$ ($l_0 = 0\%$). In this case also, all the applications react to increasing packet losses by increasing the bit-rate $r(t)$. By doubling its received data rate for $l(t) > 35\%$, TVUPlayer is the most aggressive application, while PPLive shows the smallest increase. Looking at the corresponding number of audio/video corrupted frames, it is impressive to observe that all applications achieve very good video quality up to $l(t) < 25\%$. In particular, it is worth noticing that SopCast can cope with 25% packet loss probability with only about 100 kbps of additional data rate. TVUPlayer exhibits similar performance, but at a much higher cost that account to up to 600kbps of additional data rate.

Similar observations can be drawn by looking at the decreasing packet loss probability scenario reported in bottom plots of Fig. 2. In this scenario, $I = -5\%$, $i_0 = 40\%$, $\Delta T = 5$ minutes.

These results allow us to conclude that all applications react

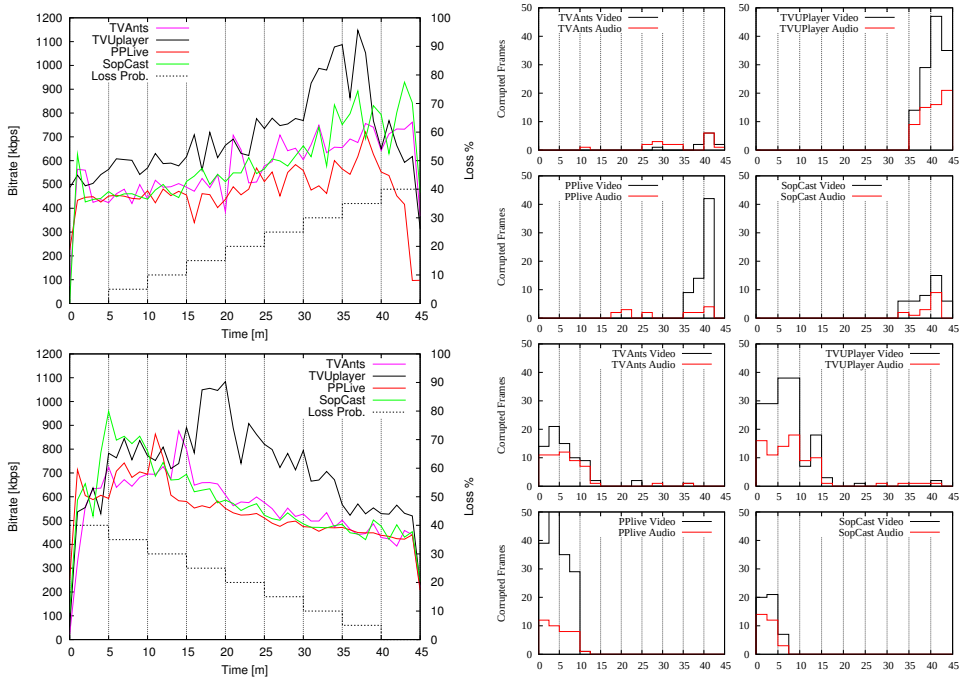


Fig. 2. Left plots: received bit-rate for increasing (top) and decreasing (bottom) loss probability. Right plots: percentage of corrupted audio/video frames, one plot for each application.

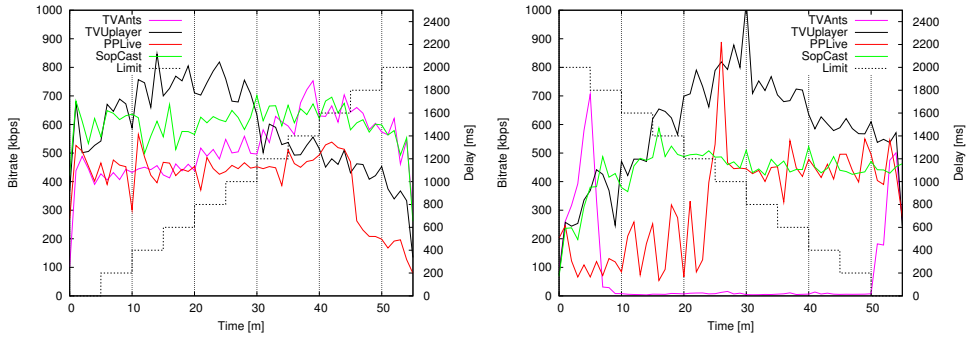


Fig. 3. Received bit-rate for increasing (left plot) and decreasing (right plot) delay.

to packet losses by trying to recover them, using some kind of ARQ mechanism that causes an increase of the received traffic. While this is very efficient in repairing the audio/video stream, it comes at the expense of an offered load that can be as large as twice the rate in normal conditions. This definitively confirms that P2P-TV applications do not perform, in general, TCP-friendly congestion control.

C. Effect of the delay

We now consider the effect of increasing and decreasing delay profiles. Fig. 3 reports the results for the received bit-rate of the increasing (left plot, $I = 200$ ms, $d_0 = 0$ ms, $\Delta T = 5$ minutes) and decreasing (right plot $I = -200$ ms, $d_0 = 2000$ ms, $\Delta T = 5$ minutes) profiles; plots about the number of corrupted frames are not reported for the sake of brevity.

Results about the increasing delay case show that the applications can manage quite well slow variations of the delay; they can stand up to 1.5 s of additional delay without any significant variation of the received bit-rate (and any audio/video error). The applications start suffering the delay when it reaches almost 2 s, which is quite large; PPLive seems the most delay sensitive application.

Interestingly, the applications seem to suffer more for large values of the delay at the start up; it is probably difficult for them to successfully create the neighbor list (notice that the additional delay applies also to packets carrying signaling information, and signaling dialogs are probably hardly completed with large values of the delay). In the decreasing profile, delay should decrease below 1.2 s to allow the applications to start receiving the video stream. Again, PPLive seems the most sensitive application: additional delay should be smaller than

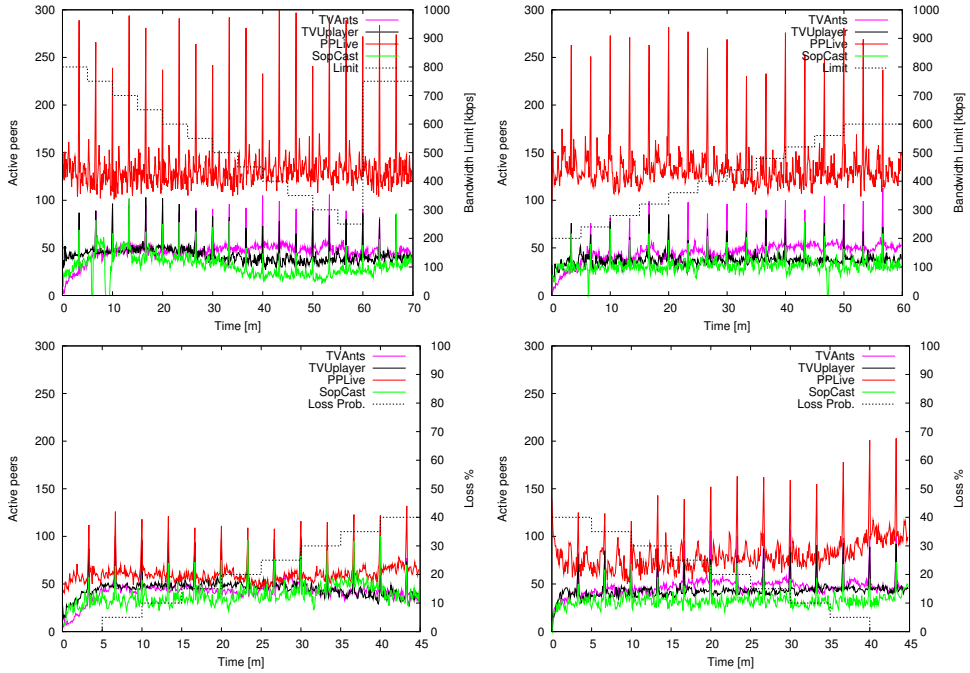


Fig. 4. Number of active peers for decreasing or increasing available bandwidth (top plots) and increasing or decreasing packet loss probability (bottom plots).

1 s to allow it to work.

D. Number of active peers

Finally, Fig. 4 reports the number of active peers, $n(t)$, for the previously described scenarios with increasing and decreasing capacity limit and loss probability. Similar results are gathered considering delay impairment.

The network conditions have no impact on the behavior of $n(t)$, which repeats regularly during the whole experiment. On the contrary, different experiments show different absolute values of $n(t)$; indeed, the absolute values change with channel popularity and time of day so as to reflect the population of available peers. PPLive, that is an extremely popular application, has always the largest values of the number of active peers.

The same observations can be made by considering the rate of new contacted peers, i.e., the number of new peers contacted in a given time interval, which is independent from the network conditions; the associated plots are not reported here for the sake of brevity. Notice also that the periodic peaks clearly visible in most of the applications are due to periodic keep-alive messages used to exchange signaling information, as already highlighted in [23].

These results allow us to conclude that the internal algorithms each application implements to discover the network, create and maintain the overlay, are completely insensitive to network conditions; network conditions influence only the video stream distribution mechanisms, i.e., the chunk scheduling.

V. Per peer impairment

A. Effect of the available capacity

We now investigate the capability of the AUT to cope with scenarios in which only a subset of peers is affected by network impairment, so that "good" and "bad" peers coexist. The goal is to verify if the AUT can identify the set of "good" peers to download from. In particular, for the results reported here, the imposed network impairment affects only peers having an odd IP address. The rational behind this choice is to have the peer population split into two equally large subsets: odd peers, affected by network impairment, and even peers, not affected.

The first row of plots in Fig. 5 reports results considering a decreasing capacity limit profile. In particular, the profile $c(t)$, that is imposed to odd peers only, starts from $c_0 = 400$ kbps, and every $\Delta T = 5$ minutes a further bandwidth decrease of $I = -25$ kbps is applied. After 60 minutes, the available bandwidth is again set to the initial value. Each plot reports, for a given application, the bit-rate received from even and odd peers and the total received bit-rate. The imposed profile is also given for completeness. Again, the applications exhibit similar behavior: during the initial phase there is no preference in receiving data from even or odd peers: they equally contribute to the total download rate. As soon as the bandwidth limit kicks in, reducing the performance of odd peers, the applications preferentially download data from even peers. The preference is stronger for PPLive and SopCast (rightmost plots) for which even peers contribute to 80-90% of download rate. TVAnts, on the contrary, adapts less than the other applications to these

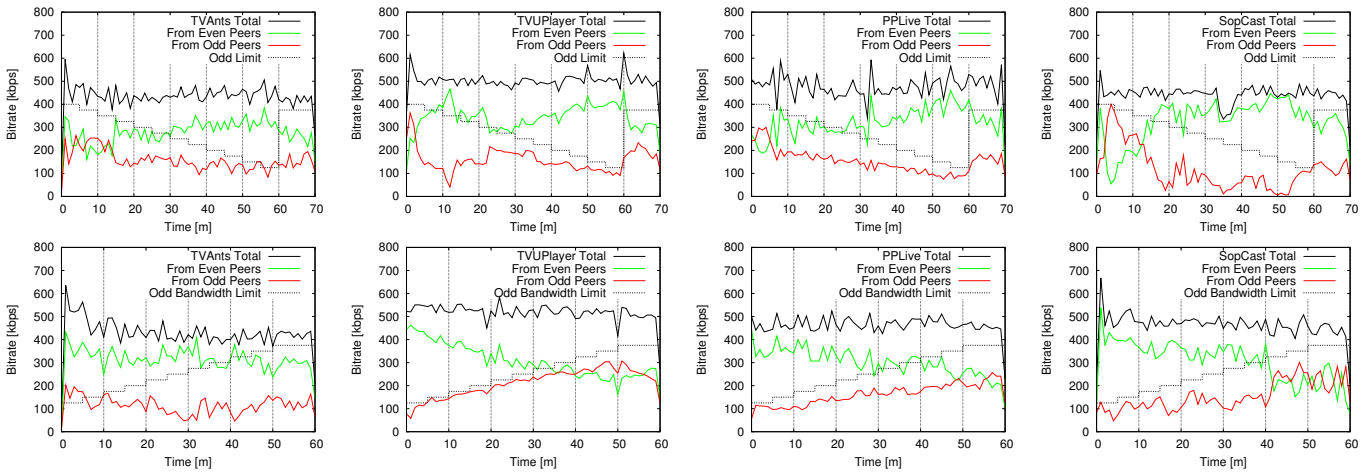


Fig. 5. Received bit-rate for decreasing (top plots) and increasing (bottom plots) capacity limit. Odd peers only are affected by the impairment.

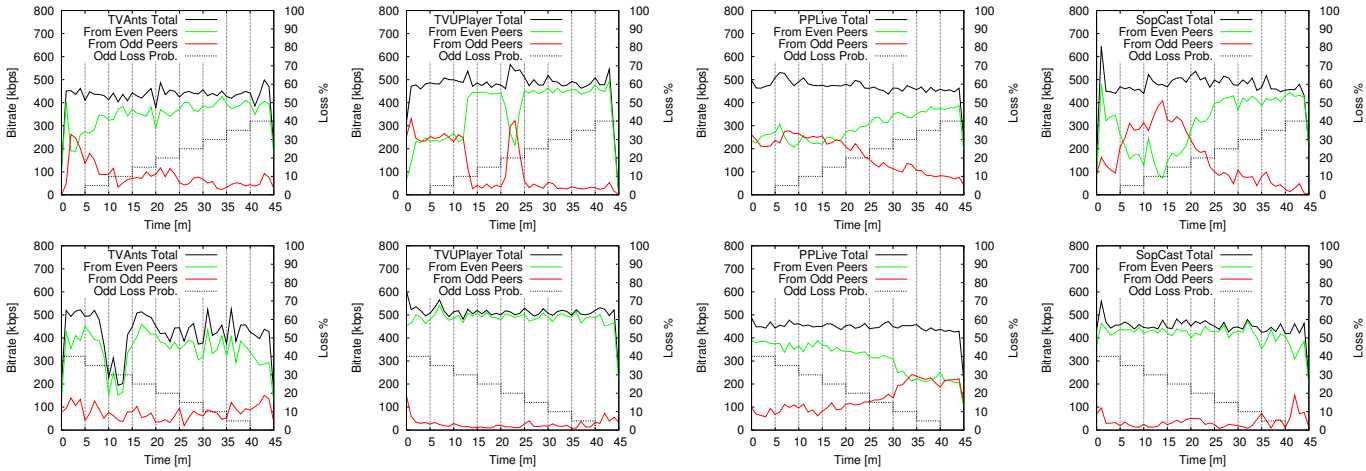


Fig. 6. Received bit-rate for increasing (top plots) and decreasing (bottom plots) loss probability. Odd peers only are affected by the impairment.

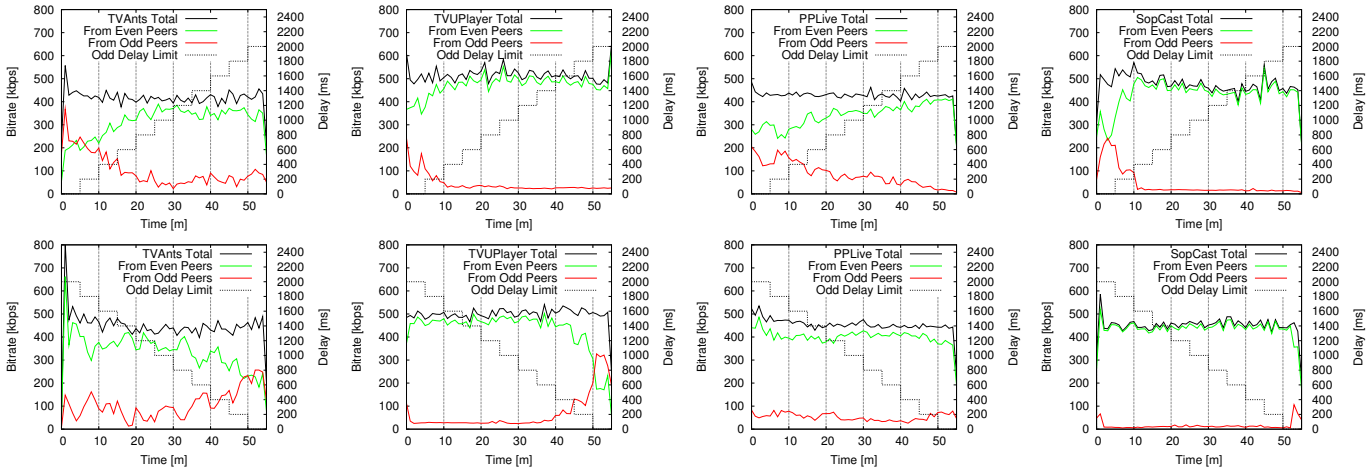


Fig. 7. Received bit-rate for increasing (top plots) and decreasing (bottom plots) delay. Odd peers only are affected by the impairment.

network conditions.

The second row of Fig. 5 reports results for an increasing bandwidth profile applied to odd peers only; the profile has parameters: $c_0 = 125$ kbps, $I = 25$ kbps, $\Delta T = 5$ minutes. Similar considerations as before hold. All applications quickly identify the adverse capacity constraints affecting odd peers, so that even peers provide larger contribution to the total download bit-rate. In particular, TVUPlayer (second plot from the left) has a very accurate control mechanism that allows it to quickly identify the changing network conditions. PPLive and SopCast also exploit the additional bandwidth of odd peers that becomes gradually available, but a longer (and more bursty) transient phase is required. Finally, TVAnts ignores the additional bandwidth, since about 70% of traffic is received from even nodes during the whole experiment.

In all cases, all the applications receive the minimum required amount of data that guarantees them to decode the audio/video streams without suffering any error.

B. Effect of the loss probability

For packet loss probability, Fig. 6 reports results considering increasing and decreasing profiles of $l(t)$. Let us start by considering increasing loss probability; plots on top row of the figure refer to a profile with $l_0 = 0\%$, $I = 5\%$, $\Delta T = 5$ minutes. In this case, different reactions are observed. TVAnts has a strong preference to receive from even peers starting from $l(t) \geq 5\%$. TVUPlayer has an on/off behavior, so that no preference is shown until $l(t) = 10\%$, and, then, 95% of data is received from even nodes only. PPLive is ignoring the loss impairment until $l(t) \geq 20\%$, after which data are preferentially received from even nodes (but still 20% of traffic is received from odd peers even when $l(t) = 40\%$). Finally, SopCast shows a more irregular and uncontrolled behavior which causes a preference toward odd peers until $l(t) = 15\%$, after which about 90% of data is received from even nodes only.

Consider now bottom plots of Fig. 6, which report results considering a decreasing profile of $l(t)$ (with $l_0 = 40\%$, $I = -5\%$, $\Delta T = 5$ minutes). Since all applications start in very unfavorable conditions for odd nodes, most of the traffic is received from even nodes. In particular, TVUPlayer constantly receives 98% of traffic from even nodes only, even when $l(t)$ becomes small. Similarly, TVAnts and SopCast exhibit a very stable preference during the whole test duration, with TVAnts trying to received 15-20% of traffic from impaired peers. PPLive, on the contrary, keeps on receiving 20% of traffic from odd nodes, percentage that goes up to 50% when $l(t) \leq 10\%$. This confirms that PPLive is capable of coping with high packet loss rates (as already noticed in Fig. 2), hinting to an effective FEC algorithm support.

C. Effect of the delay

Considering increasing delay impairment (top plots of Fig. 7, $d_0 = 2000$ ms, $I = 200$ ms, $\Delta T = 5$ minutes), we observe that all applications are very delay sensitive. In particular, SopCast and TVUPlayer peers essentially receive data only from even peers, while very little traffic is exchanged with odd peers. PPLive and TVAnts are less biased, so that additional delay of 400 ms is better tolerated.

When considering decreasing delay profile (bottom plots of Fig. 7, $d_0 = 2000$ ms, $I = -200$ ms, $\Delta T = 5$ minutes), it can be noted that the initial impaired conditions are immediately detected by all applications. Also in this case, TVUPlayer and SopCast are almost ignoring odd peers, since more than 95% of data is received from even nodes. TVAnts on the contrary keeps trying to download data from odd peers, so that they are quickly selected when $d(t) < 400$ ms. Notice that TVUPlayer shows the fastest control algorithm that allows it to receive data from odd peers once $d(t) = 0$ ms.

We performed other similar tests, targeting with impairment: a particular peer, IP subnetworks, and Autonomous Systems. All the experiments showed consistent results. We, thus, conclude that all applications implement a per-peer preference mechanism that is used to select the subset of good performing peers. While internal algorithms are unknown, the presented results suggest the applications are using different algorithms.

D. Number of active peers

Finally, in order to observe if the AUTs implement any sort of control on the number of contacted peers in presence of impairment that affects only a subset of peers, Fig. 8 reports the number of active peers distinguishing between even peers (positive y-axis) and odd peers (negative y-axis), for increasing and decreasing profiles of capacity limit, loss probability and delay; impairment applies to odd peers only. Results clearly show that the AUT keeps contacting odd peers, since $n(t)$ is not correlated with $c(t)$, $l(t)$ or $d(t)$. This hints to control algorithms that react to different network scenarios by carefully selecting the good peers to exchange data with. However, signaling is exchanged with all peers (including "bad" peers) independently from the instantaneous end-to-end network quality. We also verified that all AUTs keep exchanging data with and probing "bad" peers even during very unfavorable conditions. In these cases only few, small packets are sent and (possibly) received. This clearly indicates that only signaling information is exchanged between any two peers that are experiencing adverse network conditions, but the bad peers are not dropped in favor of the good one.

VI. Conclusions

In this paper, we propose an experimental methodology to investigate the behavior of P2P-TV applications under adverse network conditions. Since most of the successful P2P-TV applications rely on a closed and proprietary design, it is indeed important to understand if these applications implement smart algorithms to cope with different and variable network scenarios. In particular, available bandwidth, delay and packet loss probability are the most important impairment applications face in the Internet. We therefore explored how P2P-TV applications react to those parameters, by setting up real test-bed experiments. We applied this methodology to

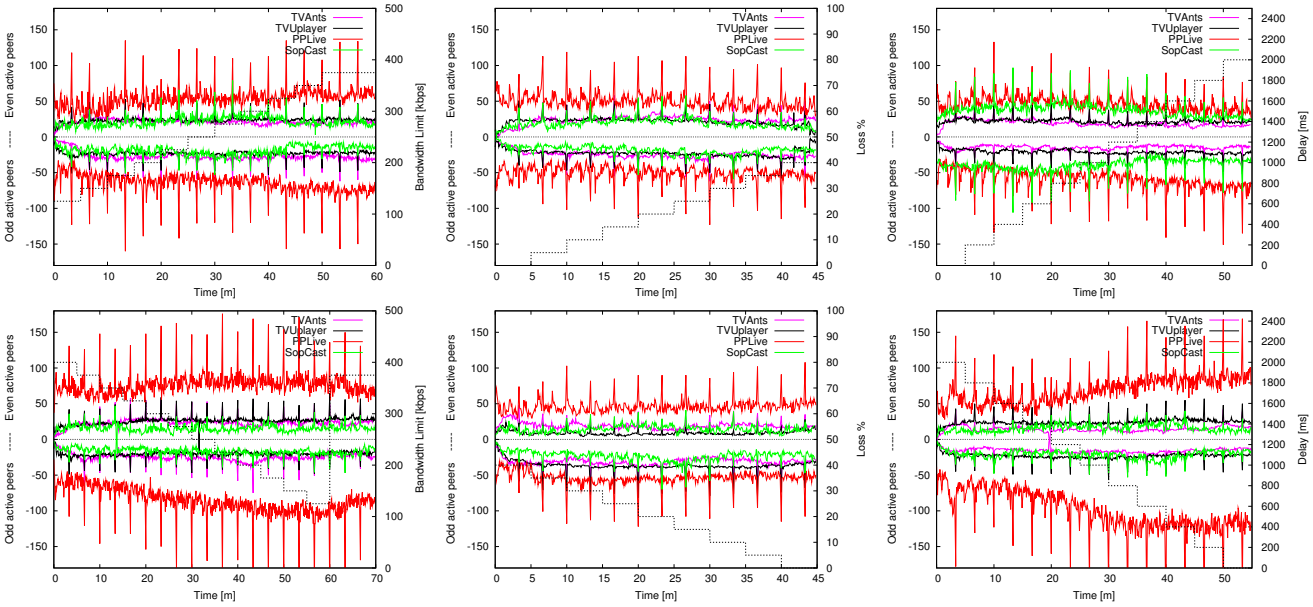


Fig. 8. Number of active peers for decreasing and increasing capacity limit (leftmost plots), increasing and decreasing packet loss probability (plots in the middle) and increasing and decreasing delay (rightmost plots). Odd peers only are affected by the impairment.

four P2P-TV applications, namely, PPLive, SopCast, TVAnts and TVUPlayer. By observing the received bit-rate and the number of contacted peers, we have shown that all applications effectively react to impairment caused by i) lack of bandwidth, ii) packet loss probability, or iii) large delay. Applications indeed successfully select the subset of peers that offer the best performance, disregarding peers on impaired paths. However, in case the bottleneck affects all peers, e.g., it is at the access link, their behavior results rather aggressive, and potentially harmful for both other applications and the network. Interestingly, the control algorithm preferentially operates by selecting the active peers among the neighbors on the overlay, but it does not affect the neighborhood selection, i.e., the overlay topology discovery and setup.

Finally, even if all applications show similar behaviors, some differences arise: TVUPlayer is the fastest and most prompt to react to changing conditions, but sometimes its control algorithm overreacts to dynamic situations; TVAnts on the contrary shows a less controlled behavior, which causes the highest overload when resources are scarce, and forces the client to keep downloading from impaired peers.

REFERENCES

[1] PPLive, http://www.pplive.com.
[2] SopCast, http://www.sopcast.com.
[3] TVAnts, http://www.tvants.com.
[4] TVUnetworks, http://www.tvunetworks.com.
[5] Joost, http://www.joost.com.
[6] Babelgum, http://www.babelgum.com.
[7] Zattoo, http://www.zattoo.com.
[8] J. Liu, S.G. Rao, B. Li; H. Zhang, "Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast," *Proceedings of the IEEE*, Vol.96, no.1, pp.11-24, Jan. 2008.
[9] A. Murray-Watson, "Internet groups warn BBC over iPlayer plans," *The Independent*, 12 August 2007.
[10] E.Leonardi, M.Mellia, A.Horvart, L.Muscariello, S.Niccolini, D.Rossi, "Building a Cooperative P2P-TV Application over a Wise Network: the Approach of the European FP-7 STREP NAPA-WINE", *IEEE Communications Magazine*, Vol. 46, pp. 20-211, April 2008.
[11] H. Xie, Y. . Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, "P4P: Provider Portal for Applications", *ACM Sigcomm 2008*, Seattle, WA, August 2008.
[12] D.Bonfiglio, M.Mellia, M.Meo, N.Ritacca, D.Rossi, "Tracking Down Skype Traffic", *IEEE Infocom*, Phoenix, AZ, April 2008.
[13] X. Hei, C. Liang, J. Liang, Y. Liu, K.W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, Vol.9, No.8, pp.1672-1687, Dec. 2007.
[14] B. Li, Y. Qu, Y. Keung, S. Xie, C. Lin, J. Liu, X. Zhang, "Inside the New Coolstreaming: Principles, Measurements and Performance Implications", *IEEE INFOCOM'08*, Phoenix, AZ, Apr. 2008.
[15] C. Wu, B. Li, S. Zhao, "Multi-channel Live P2P Streaming: Refocusing on Servers" *IEEE INFOCOM'08*, Phoenix, AZ, Apr. 2008.
[16] L. Vu, I. Gupta, J. Liang, K. Nahrstedt, "Measurement of a large-scale overlay for multimedia streaming" *Proc. of the 16th International Symposium on High Performance Distributed Computing,*, Monterey, CA, June 2007.
[17] F. Wang, J. Liu, Y. Xiong, "Stable Peers: Existence, Importance, and Application in Peer-to-Peer Live Video Streaming" *IEEE Infocom'08*, Phoenix, AZ, Apr. 2008.
[18] X. Hei, Y. Liu, K.W. Ross, "Inferring Network-Wide Quality in P2P Live Streaming Systems," IEEE JSAC, special issue on P2P Streaming, Vol.25, No.9, pp.1640-1654, Dec. 2007.
[19] S. Agarwal, J. P. Singh, A. Mavlankar, P. Baccichet, B. Girod, "Performance and Quality-of-Service Analysis of a Live P2P Video Multicast Session on the Internet," *IEEE IwQoS*, Enschede, NL, June 2008.
[20] S.Ali, A.Mathur, H.Zhang, "Measurements of Commercial Peer-To-Peer Live Video Streaming," *In Proc. of Workshop on Recent Advances in Peer-to-Peer Streaming*, Waterloo, ON, Aug. 2006.
[21] T. Silverston, O. Fourmaux, "Measuring P2P IPTV Systems," *ACM NOSSDAV'07*, Urbana-Champaign, IL, June 2007.
[22] A. Horvath, M. Telek, D. Rossi, P. Veglia, D. Ciullo, M. A. Garcia, E. Leonardi, M. Mellia, "Dissecting PPLive, SopCast, TVAnts", submitted to ACM Conext 2008.
[23] D.Ciullo, M.Mellia, M.Meo, E.Leonardi, "Understanding P2P-TV Systems Through Real Measurements", *IEEE GLOBECOM 2008*, New Orleans, FL, 30 November 2008.