P2P Implications on Web Surfing

Merciadri Luca
Luca.Merciadri@student.ulg.ac.be

First written: July 1, 2009
Last update: August 8, 2009

Abstract. P2P (Peer-to-Peer) technology comprises various ways to exchange information rapidly, each participant sharing a portion of his own resources ([1]). However, despite of the numerous advantages of using P2P, a real problem is often encountered when using cheap internet connections: web surfing becomes so slow that it seems impossible to reach a web page, for the P2P's user. It is especially the case when using connections with a low upload speed. The problem has also an importance, even if it is minor, when using high-speed connections (VDSL, ...), as it is also a waste of capacity.

Keywords: P2P, Peer-to-Peer, downloading.

1 Introduction

P2P (Peer-to-Peer) technology comprises various ways to exchange information rapidly, each participant sharing a portion of his own resources ([1]). Anyway, despite of the numerous advantages of using P2P, a real problem is often encountered when using cheap internet connections: web surfing becomes so slow that it seems impossible to reach a web page, for the P2P's user. It is especially the case when using connections with a low upload speed. The problem has also an importance, even if it is minor, when using high-speed connections (VDSL, ...), as it is also a waste of capacity.

2 Causes

This phenomenon is principally due to two concepts intrinsically linked to the user's P2P client:

1. The P2P client maintains lots of connections with other P2P clients and servers, for various purposes,
2. The P2P client tries to upload at the *maximum* speed it can upload, so it "shares" as much as it can with P2P's clients. This *maximum* speed is either (manually, or automatically[1]) set in the P2P client's options, or is simply the *maximum* the ISP (Internet Service Provider) allows its user to use.

Unexperimented users do not modify the DownLoad (DL)/UpLoad (UL) rules, as they are often available (only) in special panels, except for BitTorrent clients. It is also disadviced for beginners to modify them, as they appear to be "critical" settings, the efficiency of their P2P client greatly depending upon it. Most of P2P clients-servers use as *maximum* speeds the *maximum* available bandwidth (essentially for upload, download speed being sometimes limited by default, to prevent users with connections having high bandwidths from downloading too much). Thus, every user who does not modify these settings uses a client which uploads and downloads at the highest speed it can, depending on the available bandwidth.

[1] Even if in most cases, there is no speed limit for upload by default, when installing the P2P client-server, for evident reasons: by this way, clients can download your files, if needed, as fast as you can *really* upload.

3 Implications

However, when a P2P's user wants to visit a website, he needs (or will resign to his fate) to stop these downloads, if his P2P client-server is working, as it takes too much bandwidth, and engender too much connections. Regrettably, stopping these downloads results in lost connections with other P2P's users. As there are very often queues for overwhelmed servers[2], the P2P's user who simply wanted to visit a website loses his progression in others' queues. He will thus download at a potentially lower speed when using his P2P client on its next restarting, assuming it is restarted just after he visited the page that he reached instantly.

That is a pretty waste of time, and of computer resources. It should be avoided. Although a P2P client has to establish lots of connections, P2P should *not* avoid web surfing. Think about future: it is normal (*i.e.* a desirable thing) to download legal files, such as legally-downloadable TV series, or free books, using P2P, when surfing, both simultaneously, isn't it?

The problem could appear simple, but it is not the case, as the P2P community can only work if there is a bigger global upload bandwidth. Let $u_{\text{bandwidth}}$ denote the *maximum* global upload bandwidth given by all the users. Let $d_{\text{bandwidth}}$ denote the *maximum* global download bandwidth, the adjective "global" meaning that, if there are n persons downloading or/and uploading pieces of information, we have

$$u_{\text{bandwidth}} = \sum_{i=1}^{n} u_i,$$
$$d_{\text{bandwidth}} = \sum_{i=1}^{n} d_i,$$

the u_i and d_i being respective upload and download contributions of all the users to the community network. If the P2P community has (assuming[3] it has not stopped entirely; this phenomenon occuring during an infinitesimal time) a global ratio verifying

$$\frac{u_{\text{bandwidth}}}{d_{\text{bandwidth}}} < 1,$$

during a $\delta_t \ll$, it is litteraly dying during a δ_t time, as $d_{\text{bandwidth}} > u_{\text{bandwidth}}$. Such a situation, if encountered, is often perceived as not being an issue, as, if it happens, we have $\delta_t \ll$. If δ_t was progressively bigger and bigger each time it happens, more and more people would mistrust P2P, imagining the illegal files they are downloading and the ones they have downloaded previously are progressively being identified by government authorities, or by other "legal enemies," or thinking about other *scenarii* which could make them stopping their downloads, and, consequently, their uploads. It would cause P2P's death.

The principle of P2P is to have a list of files (which is better to have one hand on), previously downloaded or not, which contains files that can be uploaded to other P2P's users. The aim of this article is neither to show P2P's well-recognized interest, nor to make future predictions about their lifetime, but to show

1. that P2P has to be a community affair. If not, it cannot work,
2. and that making efforts to avoid P2P clients from litteraly consuming the whole bandwidth can be interesting.

Clearly, making P2P is as brilliant as well as Blizzard Entertainment uses it for its game, named "World of Warcraft"'s updates.

We could sum up these concepts in saying that *a compromise has to be found between an equal resources' sharing (as in any community) and a comfortable surfing*. Many solutions are available. For example, P2P

[2] Overwhelmed servers are a big percentage of P2P's servers, except when P2P servers are sharing uninteresting (for other users) files which are not much desired by the community.

[3] If it had stopped entirely, such a ratio would not be possible, as, for a given upload, there is a given download.

programs could lower their download and upload speeds when the user's browser would be requesting pieces of information from the World-Wide Web (WWW). Diminishing the number of connections they habitually manage would not be appreciable to the user, as it would result in low performance, except if priorities were correctly managed.

4 Results

Three experiments have been conducted, and we here present the *results*.

4.1 Parameters

Downloads In traditional P2P sessions, and, for an average Internet connection, the number n_d of downloads is often limited (either by the user, or by the client) to $10 - 20$. In the following results, $n_d = 15$.

Connection Parameters The Internet connection's parameters are defined as follows:

1. d_s is the *maximum theoretical download speed*,
2. u_s is the *maximum theoretical upload speed*,
3. $r := \frac{d_s}{u_s}$ is the *theoretical ratio download/upload*.

We use here a connection having $d_s = 4096\,\text{Kbps}$, $u_s = 256\,\text{Kbps}$, $r = 16$, from Belgium. No other application susceptible of Internet use is launched. This connection can be qualified of "cheap," as it costs $34\,€$ by month.

Clients and Technologies Two different P2P *clients* are used: µTorrent (for BitTorrent protocol), and eMule (for eD2K protocol). For both, ports are open (resulting in "Network OK" and "High ID" respectively).

µTorrent is configured as follows:

- version 1.8, build 11813,
- *maximum* upload rate: $480\,\text{Kb}\,\text{s}^{-1}$,
- *maximum* download rate: $+\infty$,
- global *maximum* number of connections: 375,
- *maximum* number of connected peers *per* torrent: 100,
- number of upload slots *per* torrent: 4,
- use additional upload slots if upload speed is $< 90\%$,
- *maximum* number of active torrents (upload): 4.

The only parameter we changed from the configuration that µTorrent would have established for our connection is the number of downloads. It was adviced to be 15 for a 10 Mbps DL connection, 8 for a 8 Mbps DL connection, and 5 for a 1 Mbps DL connection. Using this restricted table of values, and Lagrange's interpolation, and, for[4] values of DL speeds in $\{1, 10\}$ (Mbps), the given d_n (*download numbers*) for a connection having a theoretical download speed d_s is given by

$$d_n : d_s \mapsto \frac{-d_s^2 + 21 d_s + 10}{6},$$

giving $d_n = 13$ for $d_s = 4\,\text{Mbps}$. We were thus not very far from the adviced limit of downloads: we have only two downloads of more.

eMule is configured as follows:

- version 0.48a,
- *maximum* upload rate: $60\,\text{Ko}\,\text{s}^{-1}$,
- *maximum* download rate: $+\infty$ (really set to 500000, as eMule does not support this functionality),
- global *maximum* number of connections: 400,
- *maximum* number of sources *per* file: 350.

[4] After 10 Mbps, the function has evidently an uninteresting behaviour, as it is *maximum* at $d_s = 10.5$.

Downloaded Files The *downloaded files* are legal ones, enjoy the same availability on both protocols, are not the same (as it is difficult to find same *legal* files sharing same availability on both protocols), and have a total size of 30 Go.

Uploaded Files The *uploaded files* are legal ones, enjoy the same availability on both protocols, are the same, and have a total size of 1 Go.

Accessed Website The *accessed website* is always the same: `http://www.google.com/`, and the page is said to be *accessed* only when it is completely loaded (*i.e.* all the elements of the page are displayed entirely and correctly). The Google webpage is enhanced by personal news and mails widgets, which are conserved through the experiment. The used Internet browser is always the same (Mozilla Firefox, version 3.0.12), and the webpage is launched as habitually. The webpage is entirely loaded each time, as the browser's webcache is each time made empty. The page's content is not modified during the experiment.

This page was choosen as it reflects a normal page on the Internet: more text than pictures, no sounds, no movies, and forms.

Note that during Google's homepage's loading, requests are sent to `http://clients1.google.com`, for reasons that only Google knows. For this reason, two times were measured, and values are reported in the following tables.

Measured Values It is well-established that files downloading on eD2K protocol is often slower during a while (depending on download files' availability) after the launch of the eD2K's client (*i.e.* BitTorrent protocol is faster at the beginning). This was ignored: chronometers were launched at the same time for both clients. As eMule takes 3 s of more than µTorrent to be ready, measures were adapted.

It must be taken into consideration that the measures rely on various factors that cannot be simultaneously controled, such as only one is modified at a time, in a purely scientific way. Thus, despite of the attention given to these experiments and measures, results can sometimes be unprecise.

We now give Time To Access Web Pages (TTAWP) when only one client is launched and when both clients are launched, depending on Time From Client Launch (TFCL), denoted by t_l, resp. at Tables 1 and 2, p. 10 and 10. The data is plotted at Figure 1, p. 5.

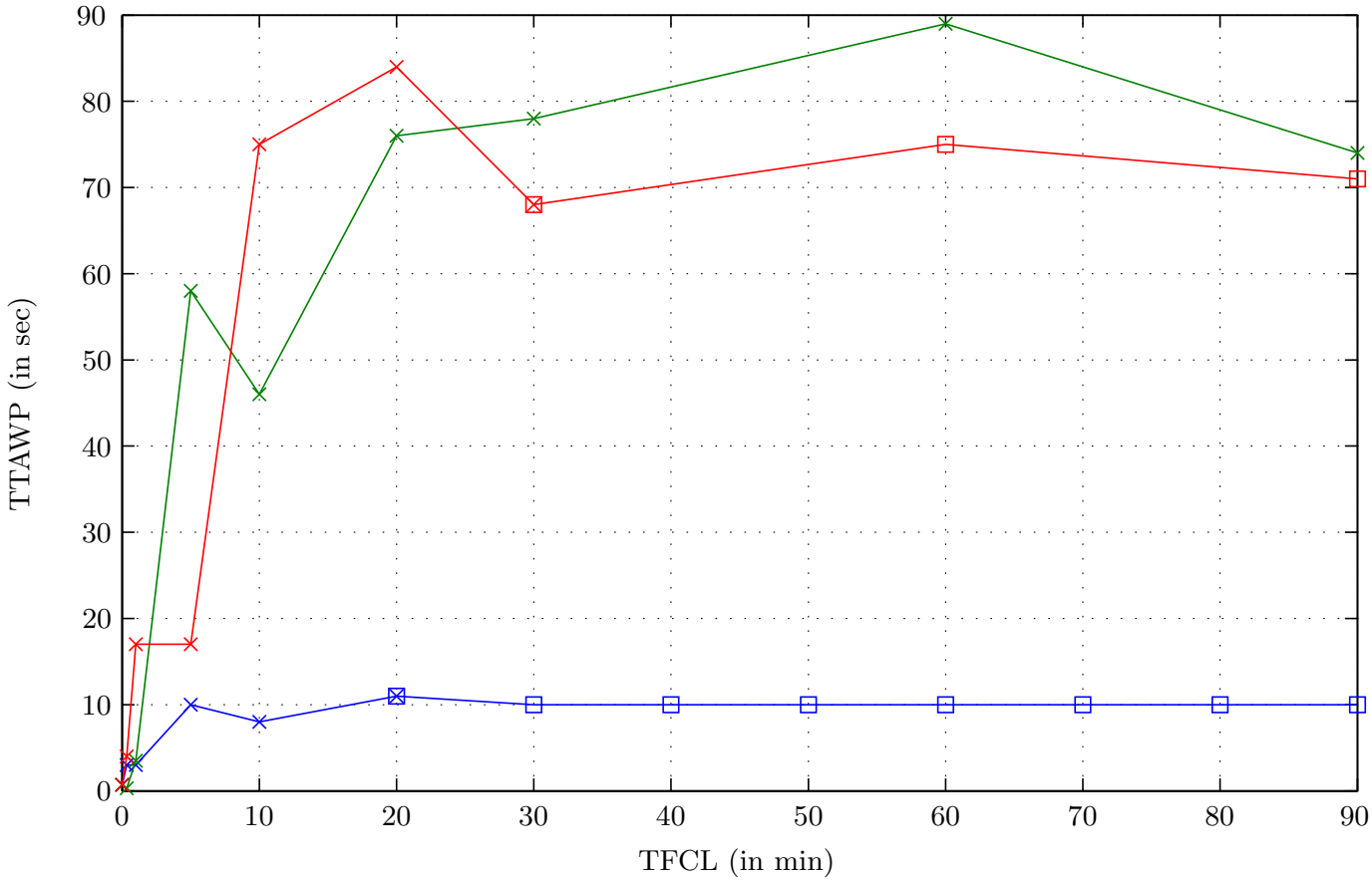

Fig. 1. Results of the data Tables 1 and 2. In blue: eMule; in green: µTorrent; in red: both. Extrapolated results are square-surrounded.

4.2 Errors

As the human reaction time equals here 1 s, and that it can vary with a term $\pm\Delta$, $\Delta = 0.5$ s, the biggest difference between a real time t_r and the measure of this same time t_m can be up to $1 + 0.5 = 1.5$ seconds; that is, 1.5 s.

Absolute Error As the biggest t_m we could find, using a given t_r, would be $t_m = t_r + 1.5$, the absolute error is given by

$$t_m - t_r = 1.5.$$

The absolute value of the absolute error verifies

$$1.5 \approx \frac{1}{2} \cdot 10^0,$$

so that there are no correct decimals. That is why we did not give any decimals in the results given above.

Very Low-Time Measures The biggest t_m we could find, using $t_r = 1\,\mathrm{s}$, would be $2.5\,\mathrm{s}$. The relative error is thus given by

$$\frac{2.5 - 1}{1} = 1.5,$$

and verifies

$$1.5 \leq \frac{1}{2} \cdot 10^{1-x},$$

iff[5] $x \leq 1 - \log 3$. Thus, t_m has $1 - \log 3 \approx 0$ significative chiffers. That is not a problem, as we were here concerned with $t_r = 1\,\mathrm{s}$, and that there is only one measure below this thresold: $0\,\mathrm{s}$, which just gives an idea about the connection's speed.

Normal-Time Measures According to the same statements, the biggest t_m we could find, using $t_r = 5\,\mathrm{s}$, would be 6.5. That is, the relative error is now given by

$$\frac{6.5 - 5}{5} = 0.3,$$

and verifies

$$0.3 \leq \frac{1}{2} \cdot 10^{1-x},$$

iff[6] $x \leq 1 - \log(0.6)$. Thus, t_m has $1 - \log(0,6) \approx 1$ significative chiffers. That is not a problem, as we were here concerned with $t_r = 5\,\mathrm{s}$: taking, for example, $t_r = 60\,\mathrm{s}$, would lead in, the worst cases, a $t_m = 61.5\,\mathrm{s}$, giving a relative error of

$$\frac{61.5 - 60}{61.5} \approx 0,$$

and verifies

$$0 \leq \frac{1}{2} \cdot 10^{1-x},$$

whatever $x \in \mathbb{R}$. Thus, t_m has a number of significative chiffers which is sufficiently high for our measures. Our measures can thus be considered as sufficiently precise for their application from 1 min.

4.3 Discussion

Remark Our aim is neither to compare eMule and µTorrent applications nor to compare eD2K and BitTorrent protocols. Both applications and protocols are interesting, for reasons which are not in the scope of this paper. We are here focused on the webpage's time to load.

[5] It can be stated easily, as $1 - \log 3 \approx 0.52$, and if we take $x \leq 0.52$, e.g. $x = 0$, we have

$$1.5 \leq \frac{1}{2} \cdot 10^{1},$$

a truth. If ones takes, e.g. $x = -100$, we have

$$1.5 \leq \frac{1}{2} \cdot 10^{101},$$

which is also true.

[6] It can be stated easily, as $1 - \log(0,6) \approx 1.22$, and if we take $x \leq 1.22$, e.g. $x = 0$, we have

$$0.3 \leq \frac{1}{2} \cdot 10^{1},$$

a truth. If ones takes, e.g. $x = -100$, we have

$$0.3 \leq \frac{1}{2} \cdot 10^{101},$$

which is also true.

µTorrent Alone It is clear that, under a threshold τ_μ (here, $\tau_\mu \approx 30\,\text{min}$), the bigger t_l, the slower the displaying of the dummy webpage. After τ_μ, results are stationary. We only have two downloads of more, and downloading two less files would result in an average

$$78 - \left(\frac{78}{15} \cdot 13\right) = 78 - 67.6 = 10.4\,\text{s}$$

time gain on 78 s. We would thus have had to wait $78 - 10.4 = 67.6\,\text{s}$ at the place of 78 s for the loading of the dummy webpage. It is clearly too long. These results confirm the hypothesis that, even if client's tips (which were given by µTorrent itself) on its tuning are taken into account, the surfing speed is still too low.

eMule Alone Under a threshold τ_e (here, $\tau_e \approx 10\,\text{s}$), the bigger t_l, the slower the displaying of the dummy webpage. After τ_e, results are stationary. These results confirm the hypothesis that, even if client's tips (which were default ones) on its tuning are taken into account, the surfing speed is still too low, even if these results are better than µTorrent's ones.

Both µTorrent and eMule Under a threshold $\tau_{\mu e}$ (here, $\tau_{\mu e} \approx 20\,\text{min}$), the bigger t_l, the slower the displaying of the dummy webpage. After $\tau_{\mu e}$, results are stationary. These results confirm the hypothesis that, even if clients' tips on their tuning are taken into account, the surfing speed is still too low.

5 Proposition

The best way to surf on the Internet when downloading files *via* P2P would be to confer to P2P's clients the ability to auto-modify their settings, according to the Internet's browser's demands. P2P's clients could hear the potential Internet's browser's demands (e.g. to launch a page, or to quit) so that they modify their settings in an automatic way (e.g. to lower the number of connections, or to increase it).

Lowering *drastically* the number of connections would have a bad influence on the P2P's client's behaviour: it would be unuseful, and disturbating, as it would not download well, and would still use bandwidth which could be useful for web surfing. Clients for eD2K often keep numerous connections with other P2P's servers, even if they are very far in the Queue. It should be avoided. For example, constantly establishing numerous connections with lots of potential uploaders where your client is only in the end of their upload queue, has no interest. However, these parameters are not often taken into account, as the developers of P2P clients do not imagine either Internet connections with such a low bandwidth, or that one could dare surfing when P2P'ing.

Lowering the download speed would be uninteresting, whatever the percentage of lowering:

1. If the download speed is lowered drastically, the P2P's client has no interest,
2. If the download speed is not lowered drastically, the P2P's client still needs to send sufficiently to contribute to the community's wellbeing.

Lowering the upload speed would be interesting, but a few remarks have to be done:

1. If it is lowered too much (using a percentage to compare with the past upload speed), and that everybody does the same, the community will crash, as exposed in Section 3, from p. 2,
2. If it is slightly lowered, it will anyway have an impact on the community, but it is not a problem if the DL speed is also lowered: people will share less, but download less, and surf more when P2P'ing.

It must be taken into account that the upload speed is often the lowest speed between DL and UL in commercial Internet solutions. As a result, even if the P2P's client does neither download too much, nor establish too much connections, the use of a big percentage (e.g. $> 70\%$) of the upload speed makes web surfing very slow.

Thus, when modifying parameters of his P2P's clients, one has to think constantly about:

1. What he wants to receive from P2P, and thus
2. What he wants to share with P2P, and thus
3. How to find a compromise between surfing as much as it is needed, and P2P'ing as much as he wants.

This is the only solution to follow until the P2P clients developers modify the in-depth foundations of their programs. We hope it will be done soon, for everybody's sake.

References

1. Schollmeier, Rdiger, *A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications*, Proceedings of the First International Conference on Peer-to-Peer Computing, (2002).

TFCL	eMule	µTorrent
0 s	< 0.7 s	
20 s	1 s or 3 s for fully charged	0.3 s
1 min	1 s or 3 s for fully charged	3.5 s
5 min	6 s or 10 s for fully charged	30 s or 58 s for fully charged
10 min	7 s or 8 s for fully charged	35 s or 46 s for fully charged
20 min	7 s or 11 s for fully charged	52 s or 76 s for fully charged
30 min	⋮	62 s or 78 s for fully charged
1 h	⋮	78 s or 89 s for fully charged
1.5 h	⋮	66 s or 74 s for fully charged
2 h	⋮	⋮
6 h	⋮	⋮

Table 1. TTAWP for eMule, and µTorrent (each one being launched alone).

TFCL	Both
0 s	< 0.7 s
20 s	3 s or 4 s for fully charged
1 min	10 s or 17 s for fully charged
5 min	16 s or 17 s for fully charged
10 min	67 s or 75 s for fully charged
20 min	74 s or 84 s for fully charged
30 min	58 s or 68 s for fully charged
1 h	⋮
1.5 h	⋮
2 h	⋮
6 h	⋮

Table 2. TTAWP for eMule and µTorrent (both being launched).